UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria dell'Informazione
Corso di Laura Magistrale in Ingegneria dell'Automazione

Tesi di Laurea Magistrale

Discrete Partitioning and Coverage Control for Asynchronous Gossiping Robots

Laureando:
Tuzón Márquez Luis

Relatore:
Prof. Carli Ruggero

Matricola:
1162246

Firma Studente: ……..

Firma Relatore: ...……

Anno Accademico 2018-2019
1 Aprile 2019

Content of the thesis

1. INTRODUCTION 10
1.1. Motivation and origin of the project 10
1.2. Objectives and scope of the project 10
1.3. Structure of the project 11

2. GRAPH THEORY 12
2.1. Model and problem formulation 12
2.1.1. Partition of the environment Q 12
2.1.2. Cost Functionals 14
2.1.3. Optimal Positions 16

3. COMMUNICATION PROTOCOLS AND COVERAGE ALGORITHM 17
3.1. Gossip coverage algorithm 18
3.2. Broadcast coverage algorithm 22

4. COMPUTATIONAL COMPLEXITY 28
4.1. Occupation matrices 29
4.2. Calculus of distances matrix 30
4.3. Centroid computation 31
4.4. Exchanging territories 32

5. SIMULATIONS 35
5.1. Simulation 1 35
5.1.1. Results using Synchronous Gossiping coverage algorithm 37
5.1.2. Results using Asynchronous Broadcast coverage algorithm 38
5.1.3. Comparative analysis 38
5.2. Simulation 2 39
5.2.1. Results using Synchronous Gossiping coverage algorithm 42
5.2.2. Results using Asynchronous Broadcast coverage algorithm 43
5.2.3. Comparative analysis 43
5.3. Simulation 3 44
5.3.1. Simulation 3 with 4 robots 44
5.3.2. Results using the Asynchronous Broadcast coverage algorithm with 4 robots 46
5.3.3. Simulation 3 with 6 robots 46
5.3.4. Results using the Asynchronous Broadcast coverage algorithm with 6 robots 48
5.3.5. Simulation 3 with 8 robots 49
5.3.6. Results using the Asynchronous Broadcast coverage algorithm with 8 robots 50
5.3.7. Comparative analysis 51

6. CONCLUSIONS AND FUTURE WORK ____ 53

7. BIBLIOGRAPHY ____ 54

APPENDIX ____ 55

Main Functions for the Synchronous Gossip coverage algorithm 55

Main function 55

Pairwise function 56

Main Functions for the Asynchronous Broadcast coverage algorithm 57

Main function 57

Pairwise function 59

Important Functions 61

Create occupancy matrix 61

Find the center of an occupancy matrix 61

Calculus of distance matrix 62

Exchange of territories 63

Calculus of cost functional 63

Support functions 63

Plot Functions 67

List of Figures

Figure 2.1 Two different Voronoi partitions of a uniform 10 x 10 grid. 14

Figure 3.1 Schematic representation of the protocols for a generic network of 4 robots. The arrows are red if the communication works in the direction pointed and the nodes are in red if they update their state. On the other hand, they are black if they are not being used. 17

Figure 3.2 Gossip coverage algorithm applied for 4 robots in a discretized squared 10x10 grid environment 21

Figure 3.3 Evolution of the cost functional in each robot 22

Figure 3.4 Asynchronous Broadcast coverage algorithm applied for 4 robots in a discretized squared 10x10 grid environment 26

Figure 3.5 Histogram with the number of communications for 100 simulations 27

Figure 4.1 Initial partition of 2 robots in a discretized square 15x15 grid environment 28

Figure 4.2 Final partition of 7 robots in a discretized square 20x20 grid environment with obstacles .. 29

Figure 4.3 Matrices of occupancy for robot 1 and robot 2 30

Figure 4.4 Region distance matrix for the occupancy region of the robot 2 31

Figure 4.5 Distance matrix for the whole environment of the robot 2 31

Figure 4.6 Partition $P1$ 32

Figure 4.7 Partition $P2$ 32

Figure 4.8 Comparing distances from robot 1 partition distance matrix and robot 2 environment distance matrix 33

Figure 4.9 Updated Partition $P1$ 33

Figure 4.10 Updated Partition $P2$ 34

Figure 4.11 Partition obtained with the exchange of territories 34

Figure 5.1 Initial conditions for simulation 1 35

Figure 5.2 Final partition with 3 robots 36

Figure 5.3 Final partition with 4 robots 36

Figure 5.4 Final partition with 5 robots 36

Figure 5.5 Final partition with 6 robots 36

Figure 5.6 Final partition with 7 robots 36

Figure 5.7 Evolution of the cost functional for different number of robots in simulation 1 37

Figure 5.8 Comparison of the amount of communications between the Synchronous Gossiping coverage algorithm and the Asynchronous Broadcast coverage algorithm. The purple one is using the Asynchronous Broadcast coverage algorithm and the yellow one is using the Synchronous Gossiping coverage algorithm 39

Figure 5.9 Simulation 2, initial partition 1 40

Figure 5.10 Simulation 2, final partition 1 40

Figure 5.11 Simulation 2, initial partition 2 40

Figure 5.12 Simulation 2, final partition 2 40

Figure 5.13 Simulation 2, initial partition 3 40

Figure 5.14 Simulation 2, final partition 3 40

Figure 5.15 Simulation 2, initial partition 4 41

Figure 5.16 Simulation 2, final partition 4 41

Figure 5.17 Simulation 2, initial partition 5 41

Figure 5.18 Simulation 2, final partition 5 41

Figure 5.19 Simulation 2, initial partition 6 41

Figure 5.20 Simulation 2, final partition 6 41

Figure 5.21 Evolution of the cost functional for each partition in simulation 2 42

Figure 5.22 Discretized environment in a 15x15 grid map for simulation 3 44

Figure 5.23 Simulation 3 with 4 robots, initial partition 1 44

Figure 5.24 Simulation 3 with 4 robots, final partition 1 44

Figure 5.25 Simulation 3 with 4 robots, initial partition 2 45

Figure 5.26 Simulation 3 with 4 robots, final partition 2 45

Figure 5.27 Simulation 3 with 4 robots, initial partition 3 45

Figure 5.28 Simulation 3 with 4 robots, final partition 3 45

Figure 5.29 Evolution of the cost functional for each partition. The blue one is the 1st partition, the red one is 2nd partition and the yellow one is the 3rd partition. 45

Figure 5.30 Simulation 3 with 6 robots, initial partition 1 47

Figure 5.31 Simulation 3 with 6 robots, final partition 1 47

Figure 5.32 Simulation 3 with 6 robots, initial partition 2 47

Figure 5.33 Simulation 3 with 6 robots, final partition 2 47

Figure 5.34 Simulation 3 with 6 robots, initial partition 3 47

Figure 5.35 Simulation 3 with 6 robots, final partition 3 47

Figure 5.36 Evolution of the cost functional for each partition. The blue one is the 1st partition, the red one is 2nd partition and the yellow one is the 3rd partition. 48

Figure 5.37 Simulation 3 with 8 robots, initial partition 1 49

Figure 5.38 Simulation 3 with 8 robots, final partition 1 49

Figure 5.39 Simulation 3 with 8 robots, initial partition 2 49

Figure 5.40 Simulation 3 with 8 robots, final partition 2 49

Figure 5.41 Simulation 3 with 8 robots, initial partition 3 49

Figure 5.42 Simulation 3 with 8 robots, final partition 3 49

Figure 5.43 Evolution of the cost functional for each partition. The blue one is the 1st partition, the red one is 2nd partition and the yellow one is the 3rd partition. 50

Figure 5.44 Amount of communications for each team robots in partition 1. The blue one is using the Synchronous Gossiping coverage algorithm and the red one is using the Asynchronous Broadcast

coverage algorithm 51

Figure 5.45 Amount of communications for each team robots in partition 2. The blue one is using the Synchronous Gossiping coverage algorithm and the red one is using the Asynchronous Broadcast coverage algorithm 52

Figure 5.46 Amount of communications for each team robots in partition 3. The blue one is using the Synchronous Gossiping coverage algorithm and the red one is using the Asynchronous Broadcast coverage algorithm 52

Abstract

This thesis proposes distributed algorithms to automatically deploy a group of robotic agents and provide coverage of a discretized environment represented by a graph.

The algorithms presented converge to a centroidal Voronoi partition using only pairwise gossip communication between the agents. The communication protocols for the algorithms are Synchronous Gossiping communication and Asynchronous Broadcast communication. Both algorithms have been implemented by the author of the thesis in MATLAB.

This thesis proves that both algorithms make the territory ownership to converge to a pairwise-optimal partition in finite time. Additionally, some simulations are done in MATLAB proving how the Asynchronous Broadcast coverage algorithm reduces the amount of communications for large teams in large environments.

1. Introduction

1.1. Motivation and origin of the project

Nowadays coordinated networks of mobile robots are being used in different applications such as environmental monitoring or warehouse logistic. The future applications for these autonomous robotic teams will revolutionize some aspects of engineering. An example of these applications are the transport of passengers and goods or search and rescue operations.

In these applications it is asked to the robots to provide a service over a space. However, when the robots are not providing the service and are waiting for the next request it is needed to reposition them to be in the best position to respond faster to the next possible service.

Some of these studies have been done by the professor Ruggero Carli [1] [2], the relator of this thesis, about discrete partitioning and coverage control with synchronous gossip communication.

1.2. Objectives and scope of the project

This thesis deals with distributed partitioning and coverage control problems for a network of robotic agents in a non-convex environment. The distributed partitioning problem for robotic networks consists in designing control and communication laws to divide an environment into territories. This is done with the scope of optimizing a cost function that measures the quality of the partitions.

In this thesis, it is described and implemented two partitioning and coverage control algorithms which optimize the configuration of a group of agents in a discrete environment represented by a graph. The optimization of the cost function depends on the locations of the agents and the distances in the graph. The challenge comes from reducing the communication requirements, by implementing two algorithms in MATLAB for coverage and partition environments with robots and simulate them in different environments and with different teams of robots.

The first algorithm implemented is Synchronous Gossiping coverage algorithm which has been already studied and simulated in [1][2]. The second algorithm is Asynchronous Broadcast coverage algorithm, which has been implemented for this thesis with the help of the professor Ruggero Carli.

The main objective of this thesis is to demonstrate how this new algorithm can reduce the communication requirements.

1.3. Structure of the project

This thesis has been organized as follows. In chapter 2, it is formally described how to discretize an environment as a connected graph. It is discussed important properties of the setup, such as that the centroid of a robot's region always belongs to the region, or that each robot's region remains connected during the evolution of the algorithm. In chapter 3 are presented the protocol communications and are explained their respective properties and the steps for pairwise communication between robots. In chapter 4, it is explained the computational complexity, and some examples are proposed to illustrate how it works. And finally, simulations and results are shown in chapter 5 and the conclusions and future work are explained in chapter 6.

2. Graph theory

Given a team of N robots with the task to coverage a finite number of points in discretized environment and with the capability to transmit information between them, we want to partition that environment in smaller regions, in which each region will be covered by one robot. This procedure will be done by using and iterative algorithm, with the objective of minimizing a cost functional. That cost functional depends on the distances of each point of the smaller region to the assigned robot.

There are different ways to solve this problem. In this thesis we will work with gossiping and broadcast communication between robots. And in both cases, we will study how the cost functional evolves with a synchronous and asynchronous transmission of data. All the methods and definitions from this chapter are taken from [1] [2].

2.1. Model and problem formulation

Given a discretized environment, let Q be the finite set of points. If each element of Q is a location and they are connected by weighted edges, we can define an undirected graph as $G = (Q, E, w)$ with edge set $E \subset Q \times Q$ and a weight map $w : E \to R_{>0}$; we let $w_e > 0$ be the weight of edge. With the assumption that G is connected and that the edge weights are the distances between locations.

In this thesis we will discretize the environment as an occupancy grid map. Each grid cell can be a free space or an occupied space. Two cells are defined as adjacent if they border each other in the grid map. To be able to form a weighted graph, each robot will define their own free cells as vertex, and each vertex will be related to its own robot by edges. The weights of the edges will be computed as the distances of each cell to the robot location (the centroid of the region).

In a weighted graph G we can assume that there is always a standard notion of distance between nodes. This is defined as *path* in G, which is an ordered sequence of nodes such that any pair of consecutive nodes in the sequence is an edge of G. The weight of a path can be defined as the sum of the weights of all the edges in the path. So, for example, to determine the distance between two vertices i and j in G, denoted by $d_G(i, j)$, we need to compute the lowest weight path from i to j, or if there is no path from i to j, that distance is $+\infty$. To sum up, the distance between two nodes is defines as the shortest path between them. Some properties to define the distances are:

- $d_G(i, j) = 0$, if and only if $i = j$
- $d_G(i, j) = d_G(j, i)$ for any $i, j \in Q$
- $d_G(i, j)$ is finite if the graph G is connected

2.1.1. Partition of the environment Q

The environment Q will be partitioned into N regions and each subset will be covered by an individual

robot, in other words, the number of robots is equal to the number of regions N. This requires creating induced subgraphs of G and defining their distances.

Given $I \subset Q$, the subgraph induced by the restriction of G to I, denoted by $G \cap I$, is defined as the graph with vertex set equal to I and its weighted edge set contains all weighted edges of $G$where both vertices belongs to I. We set $(Q, E, w) \cap I = (Q \cap I, E \cap (I \times I), w|_{I\times I})$. The induced graph is a weighted graph with a notion of distance between nodes. Which means that, given two points i and j from I, we can write that $d_I(i,j) := d_{G\cap I}(i,j)$. What's more, $d_I(i,j) \geq d_G(i,j)$.

The partition of Q is done by connected subsets. A subset can be defined as $P \subset Q$ such that $P \neq \emptyset$ and $G \cap P$ is connected. Once we have introduced that concept we can do the partition of Q into connected subsets.

Given the graph $G(Q) = (Q, E, w)$, Q can be partitioned into a collection of N connected subsets, defined as $P = \{P_i\}_{i=1}^{N}$ such that:

I. $\bigcup_{i=1}^{N} P_i = Q$;
II. $P_i \cap P_j = \emptyset$ if $i \neq j$;
III. $P_i = \emptyset$ for all $i \in \{1, \dots, N\}$;
IV. P_i is connected for all $i \in \{1, \dots, N\}$.

Let $Part_N(Q)$ to be the set of connected $N-$ partitions of Q. Deepen in the properties we can conclude that for each partition P_i that conform Q there will be a coverage of one robot, and each element of Q belongs to just one P_i (property I, II), or to the i-th robot. Each robot can only move inside of its partition, so no partition can be empty, since each partition has at least one element, the position of the robot (property III). Finally, each $P_i \in P$ induces a connected subgraph in $G(Q)$ (property IV).

It is necessary to introduce the notion of adjacent subgraphs for our communication algorithms. Two distinct connected subgraphs P_i and P_j are said to be *adjacent* if there are two vertices q_i and q_j, respectively, belonging to P_i and P_j such that $(q_i, q_j) \in E$. From that, we can observe that if P_i and P_j are adjacent then $P_i \cup P_j$ is connected. In other words, two robots i and j are adjacent or are neighbors if their subgraphs P_i and P_j are adjacent. According to this definition we can introduce an interesting and useful notion called *adjacency graph*, which is defined between regions as $\mathcal{G}(P) = \big(\{1, \dots, N\}, \mathcal{E}(\mathrm{P})\big)$, where $(i,j) \in \mathcal{E}(\mathrm{P})$ if P_i and P_j are adjacent. Since $G(P)$ is always connected, then $\mathcal{G}(P)$ is also always connected.

The method to elaborate a partition of Q and its connected subsets is iterative, and each partition P_i will change during the process of finding the best partition P, in other words, in subsequent references to P_i we will often mean $G \cap P_i$, and in fact we refer to $P_i(t)$ as the dominance subgraph or region of the i-th robot at time t.

There are several ways to partition an environment. In this thesis we have considered

to use *Voronoi partition*. Given a vector of distinct points $c \in Q^N$, the partition $P \in Part_N(Q)$ is said to be a *Voronoi partition* of Q generated by c if, for each P_i and all $k \in P_i$, we have $c_i \in P_i$, and $d_G(k, c_i) \leq d_G(k, c_j), \forall j \neq i$. As it can been seen, a Voronoi partition generated by c is not unique (this can be seen in Figure 2.1)

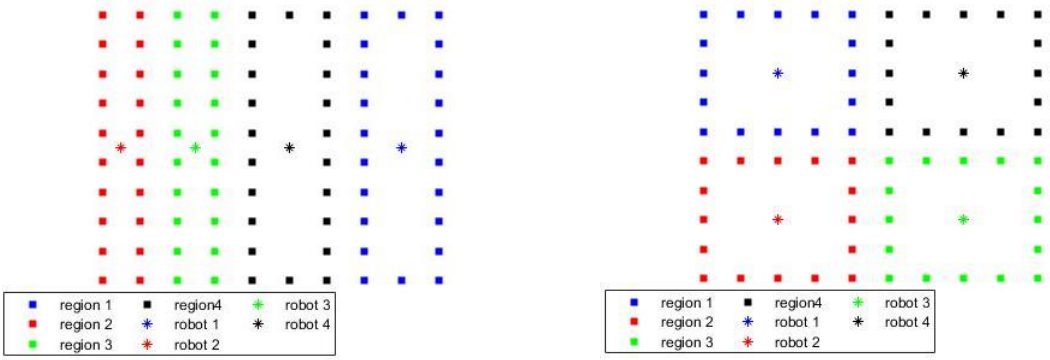


Figure 2.1 Two different Voronoi partitions of a uniform 10 x 10 grid.

The covering of Q is done by subsets that can share parts of their partitions. The main difference between covering and partition is that the property II is no more respected in case of covering.

Given the graph $G(Q) = (Q, E, w)$, Q can be covered into a collection of $\boldsymbol{N}$ connected subsets, defined as $P = \{P_i\}_{i=1}^{N}$ such that:

I. $\bigcup_{i=1}^{N} P_i = Q$;
II. $P_i = \emptyset$ for all $i \in \{1, \ldots, N\}$;
III. P_i is connected for all $i \in \{1, \ldots, N\}$.

Let $Cov_N(Q)$ to be the set of connected $N-$ partitions of Q. Deepen in the properties we can conclude that for each partition $P_i$that conform Q there will be a coverage of one robot, and each element of Q belongs to one or more than one P_i. Each robot can move inside of its partition, which includes the points shared with other partitions, so no partition can be empty, since each partition has at least one element, the position of the robot.

2.1.2. Cost Functionals

To implement the iterative method to partition Q, it is necessary to define three coverage cost functions for graphs: $H_{one}, H_{multicenter}$, and $H_{expected}$. Moreover, we define a weight function $\phi : Q \longrightarrow \mathbb{R}_{>0}$, that assigns a relative weight to each element of Q.

The first cost function H_{one} is called *one-center-function*, and its defined as:

$$H_{one}(h; A) = \sum_{k \in A} d_A(h, k)\, \phi(k)$$

If A is a connected subset contained in Q, and h is a vertex of A. Then $H_{one}(h, A)$ gives the cost for a robot to cover that subset from the vertex h, with the application of the weight function ϕ.

With the objective to minimize the one-center-function, we assume that Q is a totally ordered set, knowing that $A \in Q$, we define the set of generalized centroids of A as the set of vertices in A that minimizes H_{one}.

$$C(A) := \operatorname*{argmin}_{h \in A} H_{one}(h; A)$$

Once $C(A)$ has been defined, it is possible to define the map $C_d: C \to Q$ as $C_d(A) := \min\{c \in C(A)\}$. C_d is called the generalized centroid of A. Consequently, with this definition the centroid of a partition of Q is well-defined, and the centroid of a region will be always part of the region. From now on we will call the generalized centroid as centroid, for brevity.

To calculate the cost functional of the entire environment Q we define the *multicenter cost function* $H_{multicenter}$. Which measures the cost for N robots to cover a connected N-partition P from the vertex set $c \in Q^N$.

$$H_{muticenter}(c, P) = \frac{1}{\sum_{k \in Q} \phi(k)} \sum_{i=1}^{N} H_{one}(c_i; P_i)$$

Minimizing the performance of this cost function with respect to the vertices c and the partitions P is the main objective of our iterative algorithm. Which will give us the solution for the partition of the environment Q.

Finally, we can define the last cost function that has been introduced at the beginning of this subchapter. Let $H_{expected} : Part_N(Q) \to R_{\geq 0}$ be defined by:

$$H_{expected}(P) = H_{multicenter}(Cd(P), P)$$

In the real scenario that we are considering, each robot is asked to perform a task somewhere in its region. These tasks will appear depending on the weight function ϕ. So, by partitioning G in order to minimize the multicenter function the robot would minimize the expected distance between a task and its actual position.

2.1.3. Optimal Positions

Exploiting the definition of the multicenter cost function we can write the next proposition, which is a property of the multicenter cost function. Let $P \in Part_N(Q)$ and $c \in Q^N$. If P' is a Voronoi partition generated by c and $c' \in Q^N$ is such that $c'_i \in C(P_i)\ \forall\ i$, then

$$H_{muticenter}(c, P') \leq H_{muticenter}(c, P), and$$

$$H_{muticenter}(c', P) \leq H_{muticenter}(c, P)$$

The second inequality is strict if any $c'_i \notin C(P_i)$.

This property of the multicenter cost function implies that if (c, P) minimizes $H_{muticenter}$, then $c_i \in C(P_i)\ \forall\ i$ and P must be a Voronoi partition generated by c. As an immediate consequence of this proposition we have that: given $P \in Part_N(Q)$, if P^* is a Voronoi partition generated by $Cd(P)$ then

$$H_{expected}(P^*) \leq H_{expected}(P)$$

These statements motivate the following definition: $P \in Part_N(Q)$ is a *Centroidal Voronoi Partition* of Q if there exists a $c \in Q^N$ such that P is a Voronoi partition generated by c and $c_i \in C(P_i)\ \forall\ i$.

This let us introduce the notion of *pairwise-optimal partitions.* A partition is pairwise-optimal if, for every pair of adjacent regions, one cannot find a better two-partition of the union of the two regions. This can be formally stated as follows.

$P \in Part_N(Q)$ is a pairwise-optimal partition if for every $(i, j) \in \mathcal{E}(P)$,

$$H_{one}(Cd(P_i); P_i) + H_{one}\big(Cd\big(P_j\big); P_j\big) =$$

$$\min_{a,b \in P_i \cup P_j} \left\{ \sum_{k \in P_i \cup P_j} \min\left\{ d_{P_i \cup P_j}(a, k), d_{P_i \cup P_j}(b, k) \right\} \phi(k) \right\}$$

All pairwise-optimal partitions are also a centroidal Voronoi partitions. In other words, a set of pairwise optimal partitions is a subset of the set of centroidal Voronoi partitions.

In conclusion, for a given environment Q, a pair made of a centroidal Voronoi partition P and the corresponding vector of centroids c is locally optimal, which means that $H_{expected}$ cannot be reduced by changing either P or c independently. What's more, there does not exist a two-partition $P_i \cup P_j$ with a lower coverage cost for a pair of neighboring robots (i, j). Thus, positioning a robot in its centroidal Voronoi partition minimizes the expected distance between the robot and the task that may appear randomly in the partition of environment, owned by the robot, according to the relative weights ϕ.

3. Communication protocols and coverage algorithm

This chapter deals with the change in the communication among the robots of the network. The covering and partitioning problem described in this thesis is studied with respect to different communication protocols. All the results of this thesis have been achieved creating and implementing several functions in MATLAB in order to solve both protocols and to be able to observe their behaviors.

The types of protocols analyzed are (these protocols can be seen in Figure 3.1)

1) *Synchronous gossip.* Each robot communicates with its neighbors at each instant of time t in case of synchronous gossiping. Then, the robots update their state with the information received from their neighbors and wait until an instant of time t has passed again.
2) *Asynchronous broadcast.* One robot communicates its state to its neighbors at an instant of time t, calculated using a uniform probability distribution by the robot, or when some event happens.

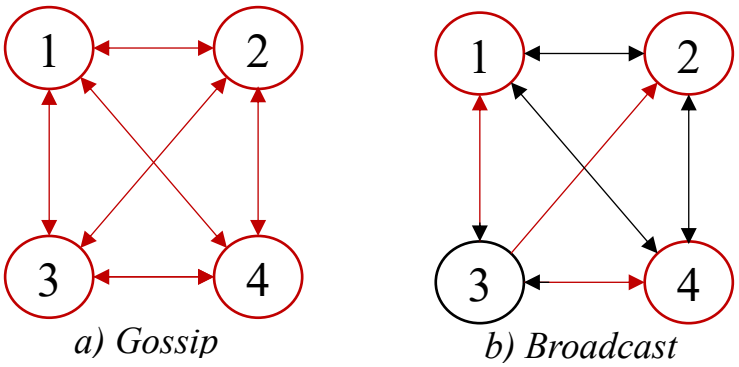


Figure 3.1 Schematic representation of the protocols for a generic network of 4 robots. The arrows are red if the communication works in the direction pointed and the nodes are in red if they update their state. On the other hand, they are black if they are not being used

As it has been said before, in gossiping communication all the nodes update their state and communicate with their neighbors every time that there is a request of communication. The study of the communication protocols is significant because it is important to compare if both systems converge to the same solution in a similar period of time. Broadcast communication permits us to reduce the number of communications in the network, which will reduce the consumption of energy of the network. As it is said in [3], the asynchronous communication permits that the algorithms can be implemented in a network composed of agents evolving at different speeds, with different computation and communication capabilities. Furthermore, convergence properties are preserved without requiring a global synchronization even if some information about neighboring robots propagates with some delay. The main advantage of asynchronism is a minimized communication overhead.

In the next subchapters of this thesis it will be explained how to perform the coverage algorithm to minimize the problem presented in chapter 2 with the communication protocols described previously.

3.1. Gossip coverage algorithm

This coverage algorithm is an implementation from the one done in the paper [1]. At each iteration, requires only a pair of adjacent regions to communicate. In order to do that, we will use the definition of *Adjacency graph*, described in chapter 2, that describe the admissible pairwise communications. The algorithm is stated as follows:

Given a group of agents $i \in \{1,\dots,N\}$, at each instant of time $t \in \mathbb{Z}_{\geq 0}$, each agent maintains in memory a connected subset $p_i(t)$. The collection $p(0) = \{p_1(0),\dots,p_N(0)\}$ is an arbitrary connected N-partition of the discretized environment Q. We have that for each instant $t \in \mathbb{Z}_{\geq 0}$ a communicating pair of robots, say $(i,j) \in E(p(t))$, is selected by a deterministic or stochastic process to be determined. Every agent $k \notin (i,j)$, sets $p_k(t+1) = p_k(t)$, while the agents i and j will perform the following tasks:

Discretized gossip coverage algorithm

i. Agent i communicates to agent j and vice-versa. Agent i sends its subset $p_i(t)$ to j and agent j sends $p_j(t)$ to i.

ii. Initialize $W_i := p_i(t)$, $W_j := p_j(t)$, $Cd(p_i(t)), Cd\left(p_j(t)\right)$

iii. When both have received the data from each other, they will compute the centroids $Cd(p_i(t+1))$ and $Cd(p_j(t+1))$ from the next sets:

$$W_{i\to j} = \left\{x \in p_i : d_{p_i \cup p_j}\left(x, Cd(p_j)\right) < d_{p_i \cup p_j}(x, Cd(p_i))\right\}$$
$$W_{j\to i} = \left\{x \in p_j : d_{p_i \cup p_j}(x, Cd(p_i)) < d_{p_i \cup p_j}\left(x, Cd(p_j)\right)\right\}$$
$$W_{i\cong j} = \left\{x \in p_i \cup p_{j_i} : d_{p_i \cup p_j}(x, Cd(p_i)) = d_{p_i \cup p_j}\left(x, Cd(p_j)\right)\right\}$$

iv. If

$$H_{one}\left(Cd(p_i(t+1)), W_{i\to j}\right) + H_{one}\left(Cd\left(p_j(t+1)\right), W_{j\to i}\right)$$
$$< H_{one}\left(Cd(p_i(t+1)), W_{i\to j}\right) + H_{one}\left(Cd(p_j(t+1)), W_{j\to i}\right)$$

v. Then

$$p_i(t+1) := \left(\left(p_i \setminus W_{i\to j}\right) \cup W_{j\to i}\right) \cup W_{j\cong i},$$
$$p_j(t+1) := \left(\left(p_j \setminus W_{j\to i}\right) \cup W_{i\to j}\right) \setminus \left(W_{j\cong i} \cap p_j\right)$$

vi. Else

$$p_i(t+1) := p_i(t)\ ;\ p_j(t+1) := p_j(t)$$

vii. End if

The cells of p_i are contained in the set $W_{i\to j}$, which are closer to $Cd(p_i(t))$, as the cells of p_j are contained respectively in the set $W_{j\to i}$, which are also closer to $Cd(p_j(t))$, whereas the set $W_{i\cong j}$ represents the set of tied cells. To summarize, when two agents or in our case two robots exchange information and by consequence territory, then their updated regions $\{p_i(t+1), p_j(t+1)\}$ are the Voronoi partition of the set $p_i(t) \cup p_j(t)$ generated by the centroids calculated on the second step $Cd(p_i(t))$ and $Cd(p_j(t))$.

We would like to define the map $T_{ij}: \mathcal{P} \longrightarrow \mathcal{P}$ for any pair $(i,j) \in \{1,\dots,N\}^2, i \neq j$ by:

$$T_{ij}(p) = \begin{cases} p, & if\ (i,j) \notin E(p) and\ W_{i\to j} \cup W_{j\to i} = \emptyset \\ (p_1,\dots,\hat{p}_i,\dots,\hat{p}_j,\dots,p_N), & otherwise \end{cases}$$

Where $\hat{p}_i = \left((p_i \setminus W_{i\to j}) \cup W_{j\to i}\right) \cup W_{j\cong i}$ and $\hat{p}_j = \left((p_j \setminus W_{j\to i}) \cup W_{i\to j}\right) \setminus (W_{j\cong i} \cap p_j)$.

Then, it is possible to say that the dynamical system on the space of all the partitions is therefore described by:

$$p(t+1) = T_{ij}(p), \qquad \text{for some } (i,j) \in E(p(t)) \text{ and } t \in \mathbb{Z}_{\geq 0}$$

Using that and the next rule, which describes what edge (i,j) is selected at each time. We can also define the set-valued map $T: \mathcal{P} \rightrightarrows \mathcal{P}$ by:

$$T(p) = \{T_{ij}(p) \mid (i,j) \in E(p(t)\}$$

With all of that, we say that our map is well-defined in the sense that $T_{ij}(p) \in P$ for all $(i,j) \in \{1,\dots,N\}^2$, if $p \in P$. This can be prove using the next technical lemma.

Let $p^+ = T_{ij}(p)$. If $p_i^+ \neq p_i$, then for all $k \in p_i^+$, every shortest path in $p_i \cup p_j$ from $Cd(p_i)$ to k is a subset of p_i^+.

Proof. Let $u = p_i \cup p_j$. This lemma can be proved by contradiction. We need to assume that, given a shortest path from $Cd(p_i)$ ot k in u, there exists $m \in Q$ which belongs both to the shortest path and to p_j^+. Considering the case in where $i < j$. For m to be in p_j^+ means that,

$$d_u(m, Cd(p_i)) > d_u\left(m, Cd(p_j)\right)$$

Which implies that

$$d_u(k, Cd(p_i)) = d_u(m, Cd(p_i)) + d_u(k,m) > d_u\left(m, Cd(p_j)\right) + d_u(k,m)$$

$$d_u\big(k, Cd(p_i)\big) \geq d_u\Big(k, Cd\big(p_j\big)\Big)$$

This is a contradiction, for $k \in p_i^+$. For the case where $i > j$ similar considerations are hold.

Now that we have proved the lemma, we can prove the fact that $T(p)$ is well-defined with the next proposition.

Let T be the pairwise algorithm map as defined before, if $p \in P$, then $T(p) \subset P$.

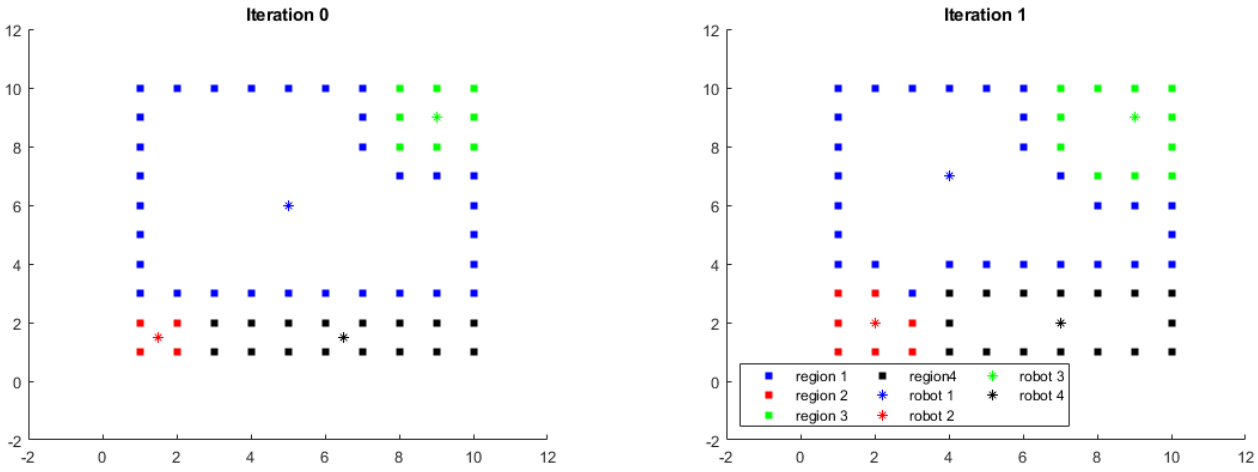
It can be observed that the previous proposition guarantees that all the regions in the partition are connected during the evolution of the algorithm. To precise, the regions are connected due to the fact that distances are taken in $p_i(t) \cup p_j(t)$. If the distances were taken in G then connectivity would no longer be hold.

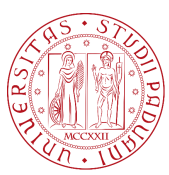
Moreover, it is worth to mention that there are several many potential variations in how to handle the tied cells in $W_{i \cong j}$. One option is to give $W_{i \cong j}$ to the agent with the lower H_{one} to achieve a form of load balancing. In any case, any method which gives $W_{i \cong j}$ to either i or j can be used.

To clarify any possible doubt from the last subchapter, we are going to present an example of how the coverage algorithm works.

The system consists in four robots in a discretized 10x10 grid square environment. Each partition for each robot has been selected randomly, and each robot has been put in the center of that partition.

In Figure 3.2., it can be seen how the robots evolve until a configuration that minimizes the functional cost.





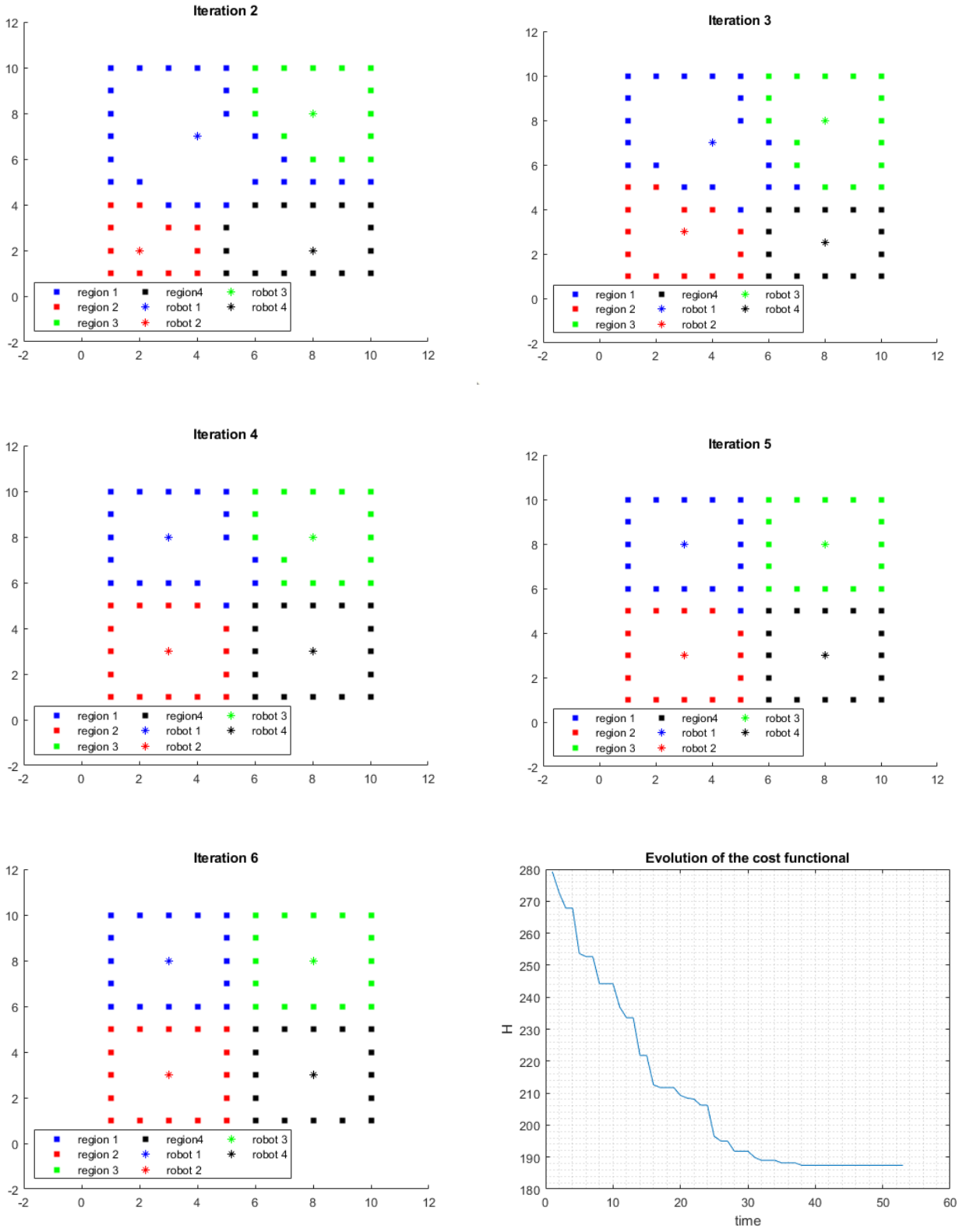


Figure 3.2 Gossip coverage algorithm applied for 4 robots in a discretized squared 10x10 grid environment

Another fact of interest would be the evolution of cost functionals, H_{one}, for each robot. As it can be observed in Figure 3.3, the different values of the function costs H_{one} evolve during the

number of iterations of the gossip coverage algorithm. The first robot, that covers almost all the environment has a high value of cost function. During the firsts iterations this cost function decreases, whereas the other cost functions of the three robots increases. Between iteration ten and eighteen, the cost functionals are increasing and decreasing, trying to converge into a similar value. Which finally occurs at the iteration number twenty. We can see that for the multicenter cost function $H_{multicenter}$ the value never increases, because of the conditions explained before, while the H_{one} of the robots can increase and decrease during the iterations of the gossip coverage algorithm, until they reach a consensus convergence value.

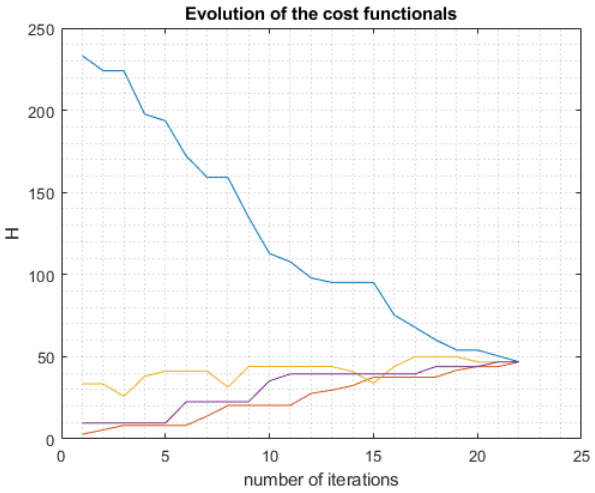


Figure 3.3 Evolution of the cost functional in each robot

3.2. Broadcast coverage algorithm

In this case, the communication is asynchronous, and the protocol communication is broadcasting. The algorithm is stated as follows:

Given a group of agents $i \in \{1,\dots,N\}$, at each instant of time $t \in \mathbb{Z}_{\geq 0}$ each agent maintains in memory a connected subset $p_i(t)$, and calculates $u_i(t) \in \{0,1\}$ following a probabilistic uniform distribution. The collection $p(0) = \{p_1(0),\dots,p_N(0)\}$ is an arbitrary connected $Cov_N(Q)$, and $u(t) = \{u_1(t),\dots,u_N(t)\}$ is a collection of numbers that provides the capability for the instant t and robot i to communicate with the other robots with the condition that $u_i(t)$ is lower than an specific value $\kappa \in \{0,1\}$, specified at the beginning of the algorithm. We have a communicating group of robots, the robot i selected by a deterministic or stochastic process to be determined, at the instant $t \in \mathbb{Z}_{\geq 0}$ and $u_i(t) < \kappa$, communicates with its neighbors. The robots will perform the following tasks:

Discretized asynchronous broadcast coverage algorithm

i. Agent i communicates to neighbor j. Agent i sends its subset $p_i(t)$ to neighbor j.

ii. Initialize $W_i := p_i(t)$, $W_j := p_j(t)$, $Cd(p_i(t)), Cd\left(p_j(t)\right)$

iii. Robot j will compute the centroid $Cd(p_j(t+1))$ from the next sets:

$$W_{j\to i} = \left\{x \in p_j \cap p_i : d_{p_i \cup p_j}(x, Cd(p_i)) \leq d_{p_i \cup p_j}\left(x, Cd(p_j)\right)\right\}$$

$$W_{i\to j} = \left\{x \in p_i : d_{p_i \cup p_j}\left(x, Cd(p_j)\right) < d_{p_i \cup p_j}(x, Cd(p_i))\right\}$$

iv. Then,

$$p_j(t+1) := \left(\left(p_j\ (t) \backslash W_{j\to i}\right) \cup\ W_{i\to j}\right)$$
$$p_i(t+1) = p_i(t)$$

v. Else

$$p_j(t+1) :=\ p_j(t)$$
$$p_i(t+1) = p_i(t)$$

vi. End if

When an agent send to another agent its information, then the robot that receives, update its region $p_j(t+1)$, which is a Voronoi partition of the set $p_i(t) \cup p_j(t)$ generated by the centroids calculated on the second step $Cd(p_i(t))$ and $Cd(p_j(t))$. However, the robot that has sent its information cannot update its state because it is not receiving any information, while the robot j could have some points of the region of the robot i or could have left some points of its region to the region i.

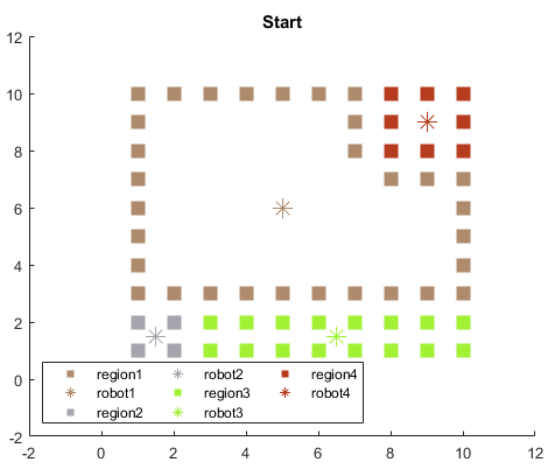
Thanks to the iterative process of communicating their states with the asynchronous broadcast protocol, all the robots end up having their regions updated and without any points shared by more than one partition of the environment.

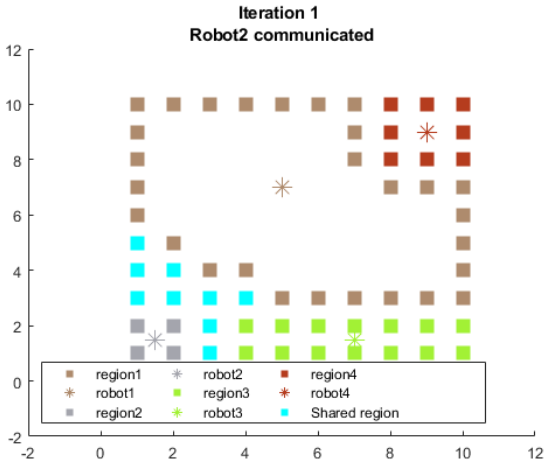
In this case the multicenter cost functional will decrease and increase due to the covering of the regions of the robots. However, the multicenter cost functional must end up having its value minimized.

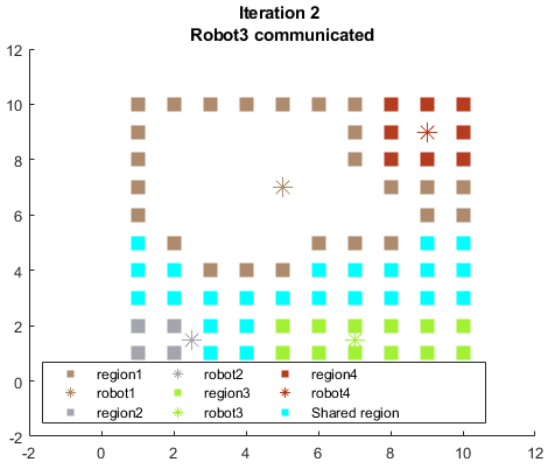
To clarify any possible doubt from the last subchapter, we are going to present an example of how the coverage algorithm works.

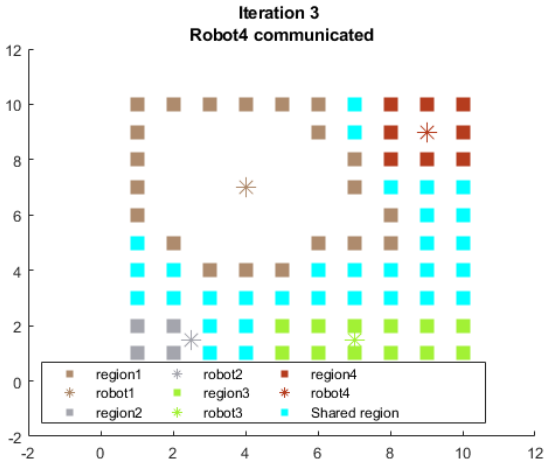
The system is the same one as the one in the discretized gossip coverage algorithm that consists in four robots in a discretized 10x10 grid square environment. Each partition for each robot has been selected randomly, and each robot has been put in the center of that partition.

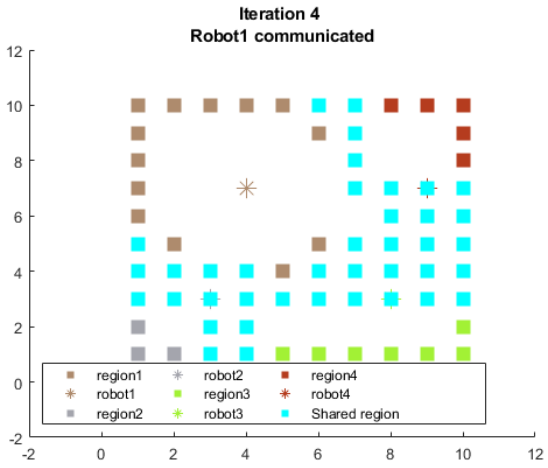
In Figure 3.4 it can be seen how the robots evolve until a configuration that minimizes the functional cost. The squares in cyan color are points shared by two or more robots, or points that must be taken by another robot.

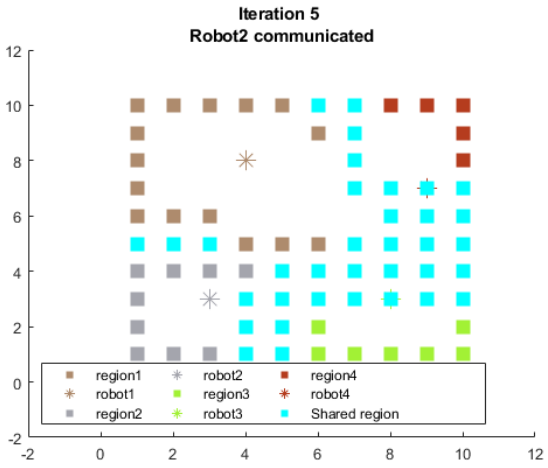

Start
region1
robot1
region2
robot2
region3
robot3
region4
robot4

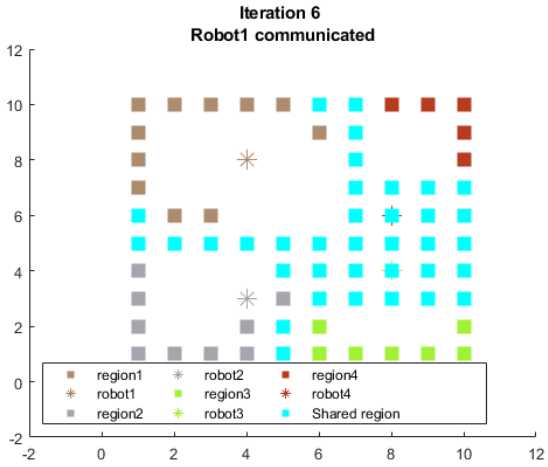

Iteration 1
Robot2 communicated
region1
robot1
region2
robot2
region3
robot3
region4
robot4
Shared region

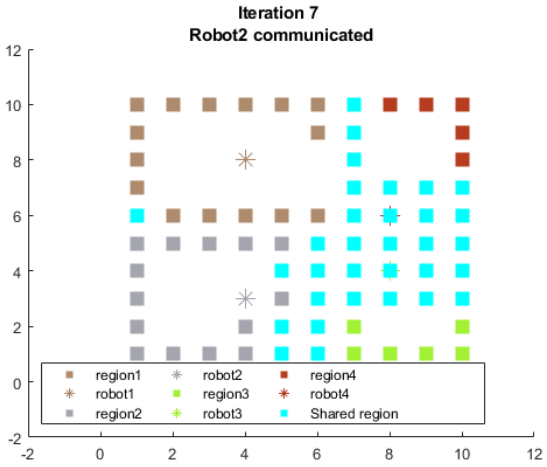

Iteration 2
Robot3 communicated
region1
robot1
region2
robot2
region3
robot3
region4
robot4
Shared region

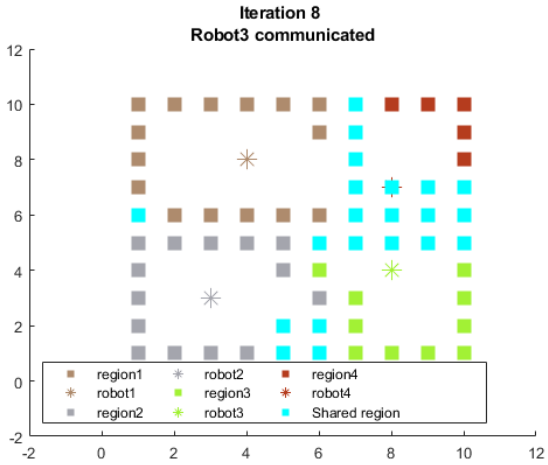

Iteration 3
Robot4 communicated
region1
robot1
region2
robot2
region3
robot3
region4
robot4
Shared region

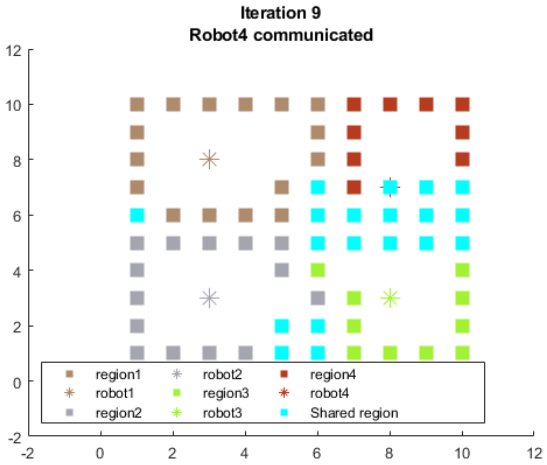

Iteration 4
Robot1 communicated
region1
robot1
region2
robot2
region3
robot3
region4
robot4
Shared region

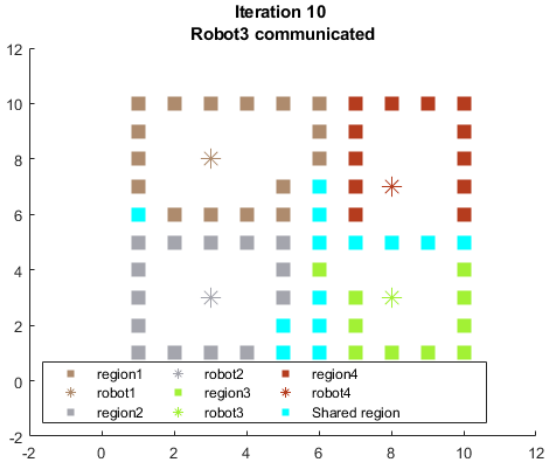

Iteration 5
Robot2 communicated
region1
robot1
region2
robot2
region3
robot3
region4
robot4
Shared region

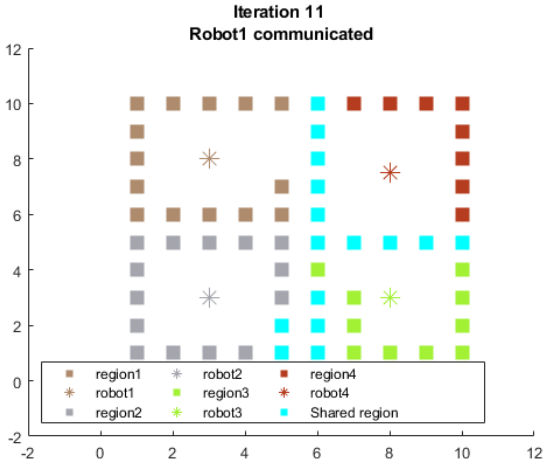

Iteration 6
Robot1 communicated
region1
robot2
region4
robot1
region3
robot4
region2
robot3
Shared region


Iteration 7
Robot2 communicated
region1
robot2
region4
robot1
region3
robot4
region2
robot3
Shared region


Iteration 8
Robot3 communicated
region1
robot2
region4
robot1
region3
robot4
region2
robot3
Shared region


Iteration 9
Robot4 communicated
region1
robot2
region4
robot1
region3
robot4
region2
robot3
Shared region


Iteration 10
Robot3 communicated
region1
robot2
region4
robot1
region3
robot4
region2
robot3
Shared region


Iteration 11
Robot1 communicated
region1
robot2
region4
robot1
region3
robot4
region2
robot3
Shared region

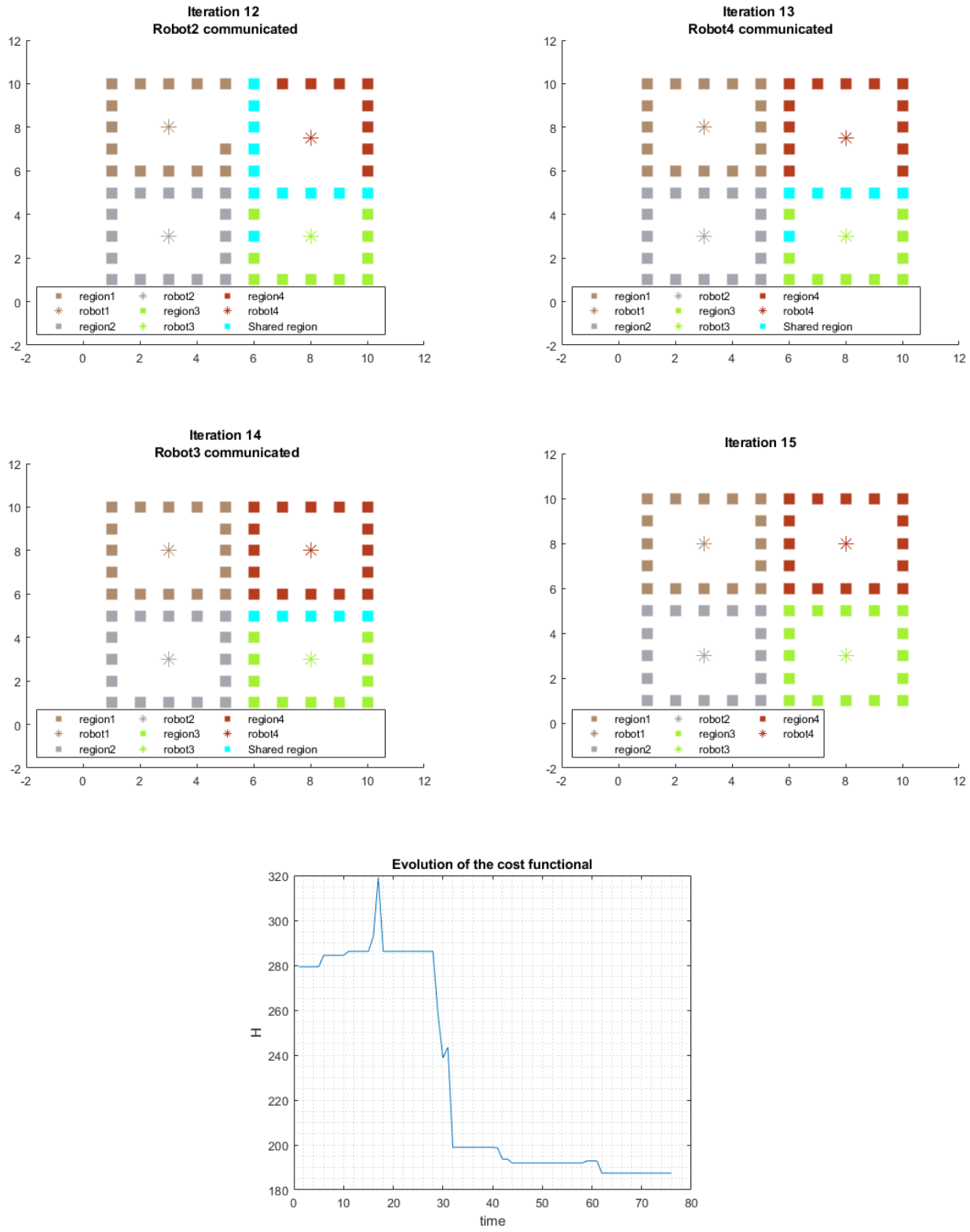


Figure 3.4 Asynchronous Broadcast coverage algorithm applied for 4 robots in a discretized squared 10x10 grid environment

It is a fact of interest to know that with this coverage algorithm we can obtain different results. The number of iterations will depend in the sequence of communications that the robots do randomly. Due to the random choice of which robots communicate, the final result for the functional cost can change in some cases, and by consequence the final partition of the environment can be also different.

In Figure 3.5 can be seen a histogram with the result of the total number of times that in the team of robots some robot communicates until they reach the final partition for 100 simulations.

As it has been shown, the number of communications related to this environment follows a Gaussian distribution, with center in 21,5. It would be interesting to compare these results with the ones obtained with gossip coverage simulation, where the number of iterations was 22. This kind of studies are done in chapter 5.

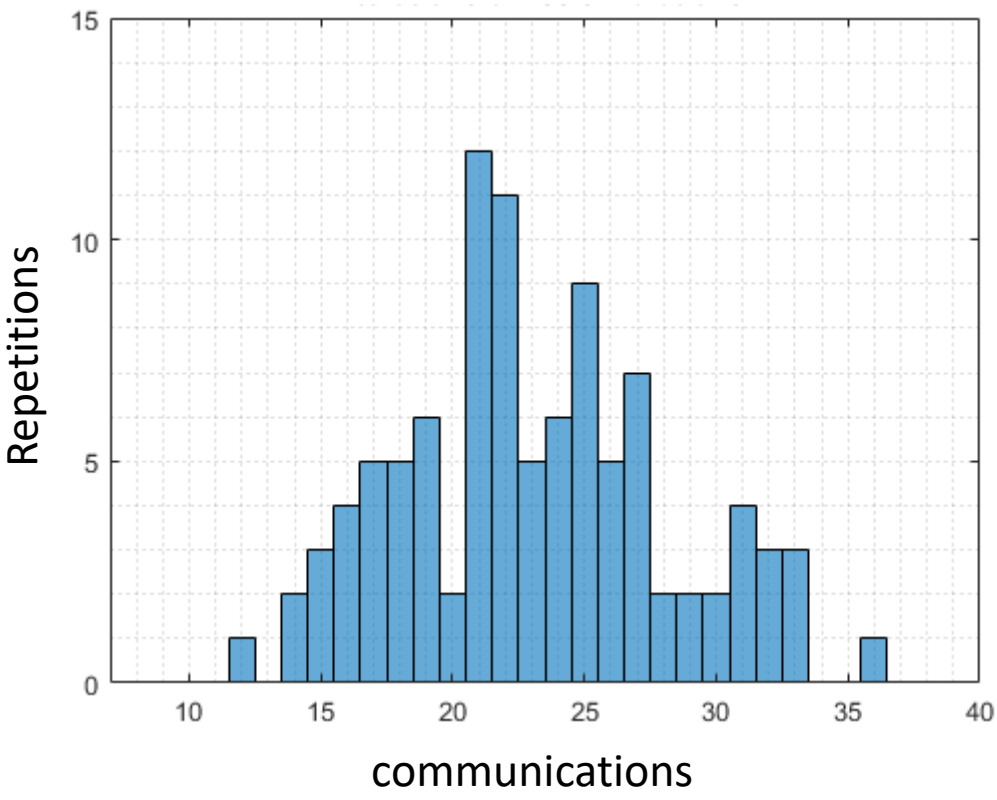


Figure 3.5 Histogram with the number of communications for 100 simulations

4. Computational complexity

In this chapter, it is described the computational requirements of the both algorithms in an environment discretized using an occupancy grid as we have used in previous chapters. Moreover, each subchapter is explained with examples in order to do easier the compression of the facts that are going to be explained. First of all, it is necessary to present the grid environment that is used in the examples. The environment is composed by a 15x15 grid and is occupied by two robots. As it is shown in Figure 4.1.

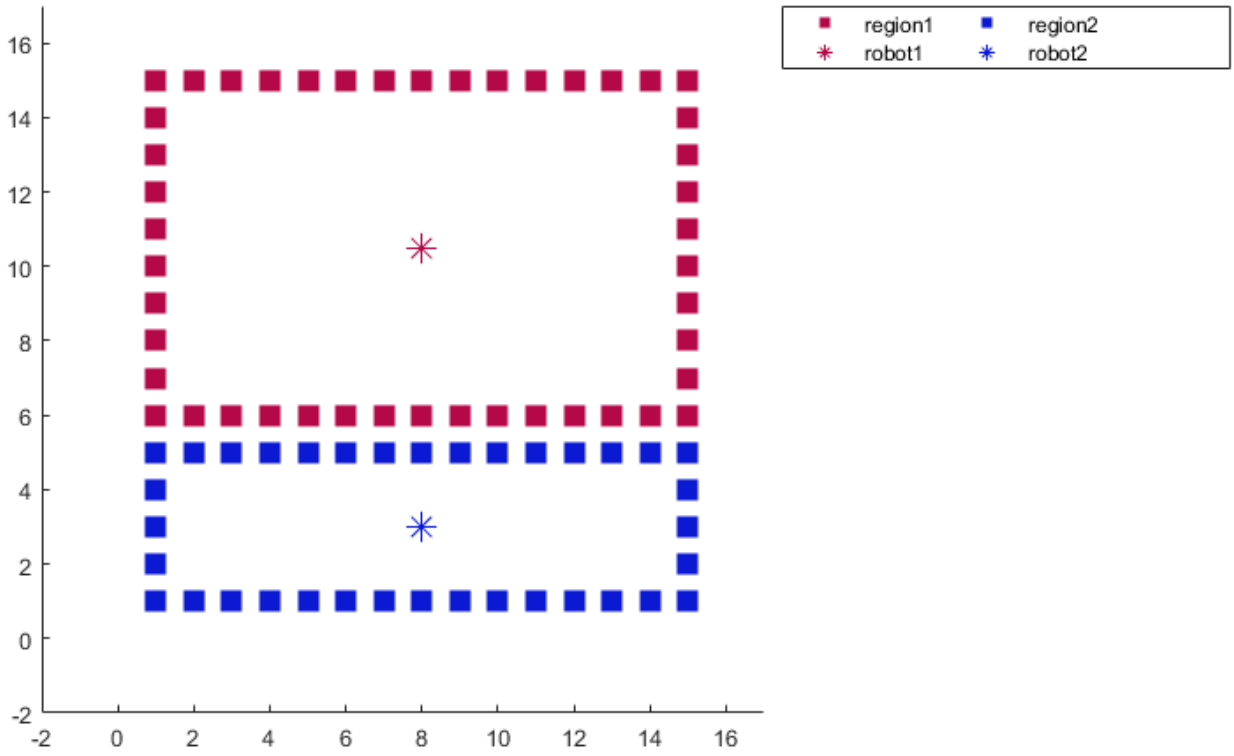


Figure 4.1 Initial partition of 2 robots in a discretized square 15x15 grid environment

Each square represents the frontier with another's robot region or the limit of the environment, and the stars represent the position of the robot on the environment. This example will be helpful to represent the different steps that the algorithms use to be able to converge into a solution.

First of all, it is necessary to explain different constraints that will affect at the time of computing the different matrix, functions and variables of the algorithm:

- The grid can be defined as a matrix $M \in \mathbb{R}^{r \times c}$, where r is a scalar and the total number of rows and c is also another scalar and the total number of columns.
- Let's define $f = \{1, 2, \dots N\}$, with N =number of robots. The region of each robot inside of the matrix is defined by f. In it can be seen how M is defined.
- Given the position of a robot such as (i, j), with $i \exists [1, r]$ and $j \exists [1, c]$. The position, which is the center of the region, can be decimal. However, when the robot creates a path then it is only able to compute using the four cardinal directions. In other words, a robot in

the position (i, j), will only be able to create a path to north $(i + 1, j)$, south $(i - 1, j)$, west $(i, j + 1)$ and east $(i, j - 1)$. Always inside the grid environment.

- A robot cannot move inside another's robot environment. The center of a Voronoi partition is the position of the robot.
- All the points of a region must be connected, at least by one square in a cardinal position. There cannot be regions of a robot that cannot not be achieved doing a path from the center of the partition to that position.
- A point can be defined as an obstacle. This kind of points cannot be taken into the regions robot and cannot be the center of a region.

In Figure 4.2, we can see the final partition of a 20x20 grid environment with 7 robots and obstacles.

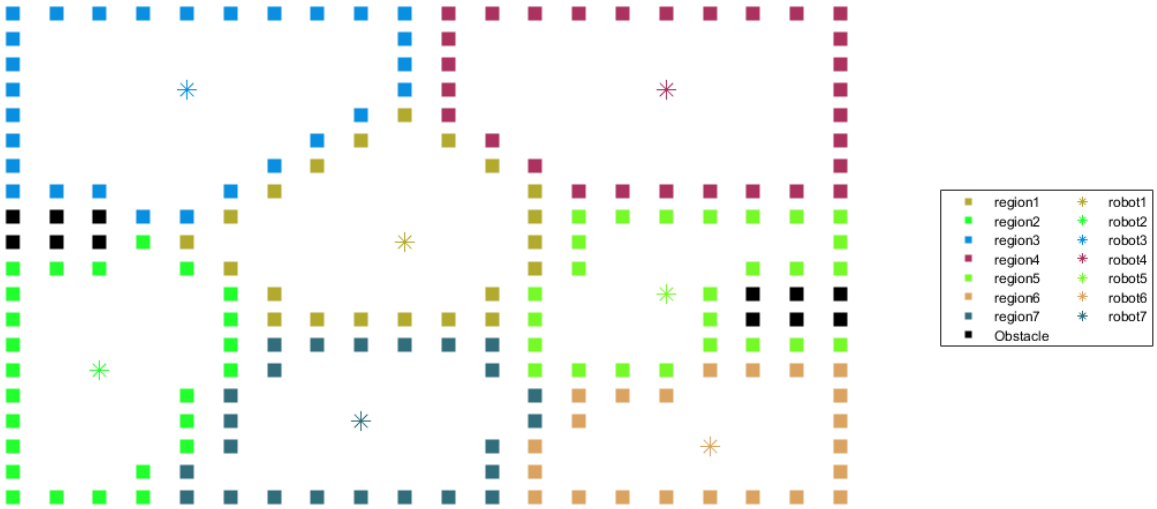


Figure 4.2 Final partition of 7 robots in a discretized square 20x20 grid environment with obstacles

As it can be seen, the robots do not interact with the points defined as obstacles, which are always represented as a black square.

4.1. Occupation matrices

For each robot in a closed environment, it is needed to define its occupation matrices. They can be defined as a copy of the grid map, it has the same size as the matrix of the grid map, but there is only the information related to the robot, obstacles and the limits of the environment.

For our example the occupation matrices of the two robots are:

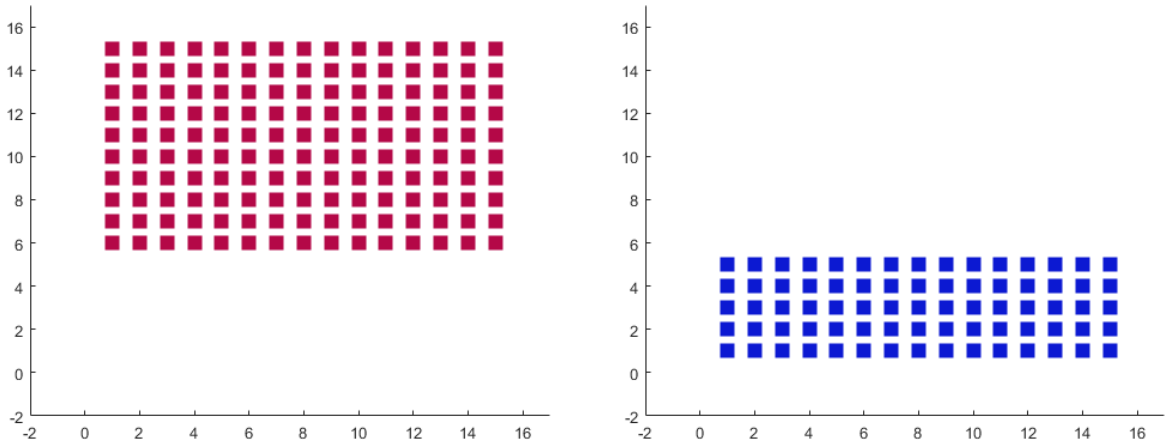


Figure 4.3 Matrices of occupancy for robot 1 and robot 2

In the occupancy matrices, the robots have all the points defined, not only the ones that make frontier with the other robots or the environment. The position of the robot, which is the center of the region is still unknown. In the next pages it is explained how to compute it.

4.2. Calculus of distances matrix

There is more than one method to compute the distances from one vertex to all other vertices in a discretized grid map of an environment Q. In this subchapter, it will be explained the method used in this thesis.

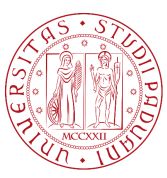
It is necessary to introduce the concept of distance matrix, which is a matrix $D \in \mathbb{R}^{r \times c}$, containing in each position the distance between the position of the robot and the respective point.

The first method is based on computing the distance as the hypotenuse between the robot and the point. Let's define the position of the robot as i and the point as j. Then:

$$d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

Using this distance, it is easy to create a distance matrix for each robot, in which each position has its respective distance to the robot.

That means that each robot has its own distance matrix. What's more, each robot calculates a distance matrix for the entire environment and another distance matrix for its occupancy region.



7.2801	7.0711	7	7.0711	7.2801	0	0	0	0	0	0	0	0	0	0
6.3246	6.0828	6	6.0828	6.3246	0	0	0	0	0	0	0	0	0	0
5.3852	5.0990	5	5.0990	5.3852	0	0	0	0	0	0	0	0	0	0
4.4721	4.1231	4	4.1231	4.4721	0	0	0	0	0	0	0	0	0	0
3.6056	3.1623	3	3.1623	3.6056	0	0	0	0	0	0	0	0	0	0
2.8284	2.2361	2	2.2361	2.8284	0	0	0	0	0	0	0	0	0	0
2.2361	1.4142	1	1.4142	2.2361	0	0	0	0	0	0	0	0	0	0
2	1	0	1	2	0	0	0	0	0	0	0	0	0	0
2.2361	1.4142	1	1.4142	2.2361	0	0	0	0	0	0	0	0	0	0
2.8284	2.2361	2	2.2361	2.8284	0	0	0	0	0	0	0	0	0	0
3.6056	3.1623	3	3.1623	3.6056	0	0	0	0	0	0	0	0	0	0
4.4721	4.1231	4	4.1231	4.4721	0	0	0	0	0	0	0	0	0	0
5.3852	5.0990	5	5.0990	5.3852	0	0	0	0	0	0	0	0	0	0
6.3246	6.0828	6	6.0828	6.3246	0	0	0	0	0	0	0	0	0	0
7.2801	7.0711	7	7.0711	7.2801	0	0	0	0	0	0	0	0	0	0

Figure 4.4 Region distance matrix for the occupancy region of the robot 2

7.2801	7.0711	7	7.0711	7.2801	7.6158	8.0623	8.6023	9.2195	9.8995	10.6301	11.4018	12.2066	13.0384	13.8924
6.3246	6.0828	6	6.0828	6.3246	6.7082	7.2111	7.8102	8.4853	9.2195	10	10.8167	11.6619	12.5300	13.4164
5.3852	5.0990	5	5.0990	5.3852	5.8310	6.4031	7.0711	7.8102	8.6023	9.4340	10.2956	11.1803	12.0830	13
4.4721	4.1231	4	4.1231	4.4721	5	5.6569	6.4031	7.2111	8.0623	8.9443	9.8489	10.7703	11.7047	12.6491
3.6056	3.1623	3	3.1623	3.6056	4.2426	5	5.8310	6.7082	7.6158	8.5440	9.4868	10.4403	11.4018	12.3693
2.8284	2.2361	2	2.2361	2.8284	3.6056	4.4721	5.3852	6.3246	7.2801	8.2462	9.2195	10.1980	11.1803	12.1655
2.2361	1.4142	1	1.4142	2.2361	3.1623	4.1231	5.0990	6.0828	7.0711	8.0623	9.0554	10.0499	11.0454	12.0416
2	1	0	1	2	3	4	5	6	7	8	9	10	11	12
2.2361	1.4142	1	1.4142	2.2361	3.1623	4.1231	5.0990	6.0828	7.0711	8.0623	9.0554	10.0499	11.0454	12.0416
2.8284	2.2361	2	2.2361	2.8284	3.6056	4.4721	5.3852	6.3246	7.2801	8.2462	9.2195	10.1980	11.1803	12.1655
3.6056	3.1623	3	3.1623	3.6056	4.2426	5	5.8310	6.7082	7.6158	8.5440	9.4868	10.4403	11.4018	12.3693
4.4721	4.1231	4	4.1231	4.4721	5	5.6569	6.4031	7.2111	8.0623	8.9443	9.8489	10.7703	11.7047	12.6491
5.3852	5.0990	5	5.0990	5.3852	5.8310	6.4031	7.0711	7.8102	8.6023	9.4340	10.2956	11.1803	12.0830	13
6.3246	6.0828	6	6.0828	6.3246	6.7082	7.2111	7.8102	8.4853	9.2195	10	10.8167	11.6619	12.5300	13.4164
7.2801	7.0711	7	7.0711	7.2801	7.6158	8.0623	8.6023	9.2195	9.8995	10.6301	11.4018	12.2066	13.0384	13.8924

Figure 4.5 Distance matrix for the whole environment of the robot 2

In Figure 4.4 and Figure 4.5 there are the distance matrices for the robot 2. In this example both matrices have been calculated using the center of the region as the point of the robot. To be able to compute the center of the robot, it is necessary to see the Subchapter 4.3.

4.3. Centroid computation

From Chapter 2 Graph theory, it is known that the centroid c_i is the vertex in the partition P_i that minimizes its cost function H_{one}, the sum of the distances to all other vertices in P_i. Then, when a robot is solving the problem to determine the vertex that will be c_i then it is solving a 1-center problem on the induced subgraph $G \cap P_i$.

The approach that has been selected to calculate the center in this thesis works calculating the distance matrix for each vertex of the occupancy matrix of the robot. Saving the vertex that has the smallest H_{one} and updating it and the distance matrix in case of finding another one that has a smaller H_{one}.

This approach finds the true centroid in the region but requires $O(|P_i|)^2$ time.

4.4. Exchanging territories

There are some steps to be able compute the pairwise computation of which territory cells to exchange between robots i and j. In case of synchronous gossip communication:

1. Exchanging their current subsets P_i and P_j between robot i and j.
2. Each agent computes the centroids c_i and c_j.
3. Computing which cells are closer to c_i and which cells are closer to c_j.
4. Updating the regions P_i and P_j.
5. Updating the centroids c_i and c_j

On the other hand, in case of asynchronous broadcast communication:

1. Robot i sends P_i to robot j.
2. Agent j computes the centroid c_i.
3. Computing which cells are closer to c_i and which cells are closer to c_j.
4. Updating the regions P_i and P_j.
5. Updating the centroids c_i and c_j.

In the particular case of our example with the synchronous Gossip communication, the algorithm would work as follows:

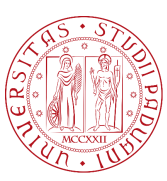
1. **Exchanging P_1 and P_2.**

0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

Figure 4.6 Partition P_1

2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	0	0	0	0	0	0	0	0	0	0

Figure 4.7 Partition P_2



2. Calculus of the centroids 1 and 2

	Centroid 1	**Centroid 2**
X	8	8
Y	10.5	3

Table 4.1 Centroids of robot 1 and robot 2

3. Computing which cells are close to robot 2.

0	0	0	0	0	8.3217	7.8262	7.4330	7.1589	7.0178	7.0178	7.1589	7.4330	7.8262	8.3217
0	0	0	0	0	7.5000	6.9462	6.5000	6.1847	6.0208	6.0208	6.1847	6.5000	6.9462	7.5000
0	0	0	0	0	6.7268	6.1033	5.5902	5.2202	5.0249	5.0249	5.2202	5.5902	6.1033	6.7268
0	0	0	0	0	6.0208	5.3151	4.7170	4.2720	4.0311	4.0311	4.2720	4.7170	5.3151	6.0208
0	0	0	0	0	5.4083	4.6098	3.9051	3.3541	3.0414	3.0414	3.3541	3.9051	4.6098	5.4083
0	0	0	0	0	4.9244	4.0311	3.2016	2.5000	2.0616	2.0616	2.5000	3.2016	4.0311	4.9244
0	0	0	0	0	4.6098	3.6401	2.6926	1.8028	1.1180	1.1180	1.8028	2.6926	3.6401	4.6098
0	0	0	0	0	4.5000	3.5000	2.5000	1.5000	0.5000	0.5000	1.5000	2.5000	3.5000	4.5000
0	0	0	0	0	4.6098	3.6401	2.6926	1.8028	1.1180	1.1180	1.8028	2.6926	3.6401	4.6098
0	0	0	0	0	4.9244	4.0311	3.2016	2.5000	2.0616	2.0616	2.5000	3.2016	4.0311	4.9244
0	0	0	0	0	5.4083	4.6098	3.9051	3.3541	3.0414	3.0414	3.3541	3.9051	4.6098	5.4083
0	0	0	0	0	6.0208	5.3151	4.7170	4.2720	4.0311	4.0311	4.2720	4.7170	5.3151	6.0208
0	0	0	0	0	6.7268	6.1033	5.5902	5.2202	5.0249	5.0249	5.2202	5.5902	6.1033	6.7268
0	0	0	0	0	7.5000	6.9462	6.5000	6.1847	6.0208	6.0208	6.1847	6.5000	6.9462	7.5000
0	0	0	0	0	8.3217	7.8262	7.4330	7.1589	7.0178	7.0178	7.1589	7.4330	7.8262	8.3217
7.2801	7.0711	7	7.0711	7.2801	7.6158	8.0623	8.6023	9.2195	9.8995	10.6301	11.4018	12.2066	13.0384	13.8924
6.3246	6.0828	6	6.0828	6.3246	6.7082	7.2111	7.8102	8.4853	9.2195	10	10.8167	11.6619	12.5300	13.4164
5.3852	5.0990	5	5.0990	5.3852	5.8310	6.4031	7.0711	7.8102	8.6023	9.4340	10.2956	11.1803	12.0830	13
4.4721	4.1231	4	4.1231	4.4721	5	5.6569	6.4031	7.2111	8.0623	8.9443	9.8489	10.7703	11.7047	12.6491
3.6056	3.1623	3	3.1623	3.6056	4.2426	5	5.8310	6.7082	7.6158	8.5440	9.4868	10.4403	11.4018	12.3693
2.8284	2.2361	2	2.2361	2.8284	3.6056	4.4721	5.3852	6.3246	7.2801	8.2462	9.2195	10.1980	11.1803	12.1655
2.2361	1.4142	1	1.4142	2.2361	3.1623	4.1231	5.0990	6.0828	7.0711	8.0623	9.0554	10.0499	11.0454	12.0416
2	1	0	1	2	3	4	5	6	7	8	9	10	11	12
2.2361	1.4142	1	1.4142	2.2361	3.1623	4.1231	5.0990	6.0828	7.0711	8.0623	9.0554	10.0499	11.0454	12.0416
2.8284	2.2361	2	2.2361	2.8284	3.6056	4.4721	5.3852	6.3246	7.2801	8.2462	9.2195	10.1980	11.1803	12.1655
3.6056	3.1623	3	3.1623	3.6056	4.2426	5	5.8310	6.7082	7.6158	8.5440	9.4868	10.4403	11.4018	12.3693
4.4721	4.1231	4	4.1231	4.4721	5	5.6569	6.4031	7.2111	8.0623	8.9443	9.8489	10.7703	11.7047	12.6491
5.3852	5.0990	5	5.0990	5.3852	5.8310	6.4031	7.0711	7.8102	8.6023	9.4340	10.2956	11.1803	12.0830	13
6.3246	6.0828	6	6.0828	6.3246	6.7082	7.2111	7.8102	8.4853	9.2195	10	10.8167	11.6619	12.5300	13.4164
7.2801	7.0711	7	7.0711	7.2801	7.6158	8.0623	8.6023	9.2195	9.8995	10.6301	11.4018	12.2066	13.0384	13.8924

Figure 4.8 Comparing distances from robot 1 partition distance matrix and robot 2 environment distance matrix

4. Update P_1 and P_2.

0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

Figure 4.9 Updated Partition P_1

2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	0	0	0	0	0	0	0	0	0

Figure 4.10 Updated Partition P_2

5. Calculating new centroids

	Updated Centroid 1	**Updated Centroid 2**
X	8	8
Y	11	3.5

Table 4.2 Updated centroids of robot 1 and robot 2

The final partition due to this exchange in territories is shown in Figure 4.11.

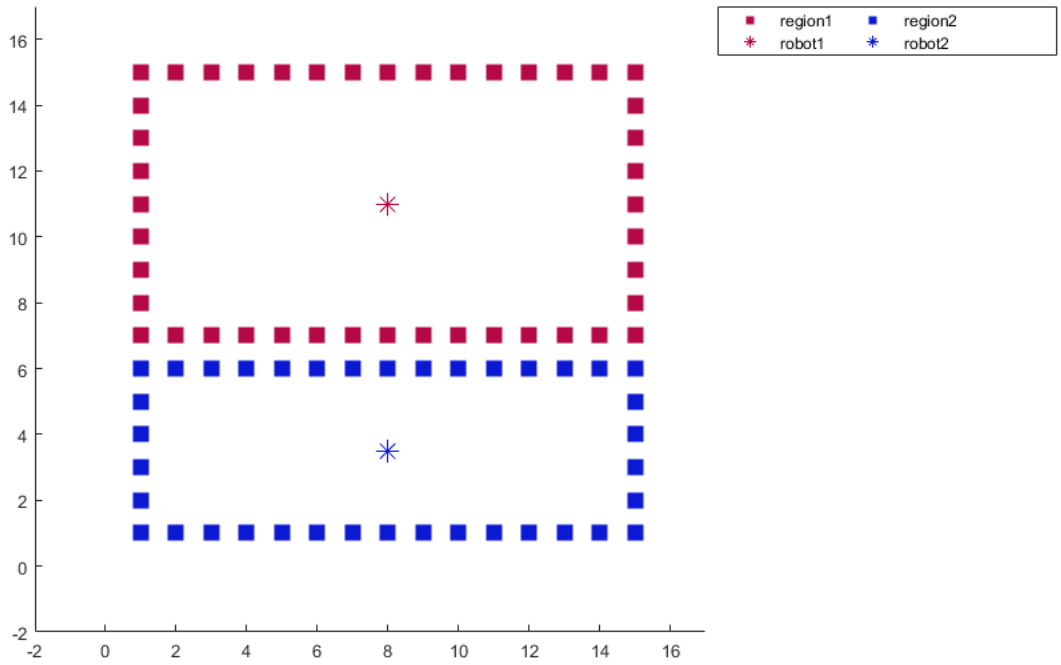


Figure 4.11 Partition obtained with the exchange of territories

5. Simulations

In this chapter it is going to be compared the results in simulation for different environments with different teams of robots, to demonstrate the utility of the Asynchronous Broadcast coverage algorithm. As in the previous chapters, for the simulations is going to be used the occupancy grid maps. All the constraints and method explained in Chapter 3 Communication protocols and coverage algorithm and Chapter 4 Computational complexity have been used to simulate. All the functions have been created in MATLAB and can be seen in the appendix.

To normalize the initial conditions in the maps, robot 1 is going to cover the whole map and the other robots will over just its own position at the beginning of each simulation

5.1. Simulation 1

The first simulation shown consists in a grid map of 22x25 with two blocks of obstacles. The teams of robots will be made from 3 to 7 robots. The initial conditions have been chosen randomly. Figure 5.1 corresponds to the initial representation of the map with 7 robots. For the simulations with less robots, the positions have taken with respect to the ones chosen in the simulation of 7 robots. To be able to show with more accuracy the map the legend has not been included.

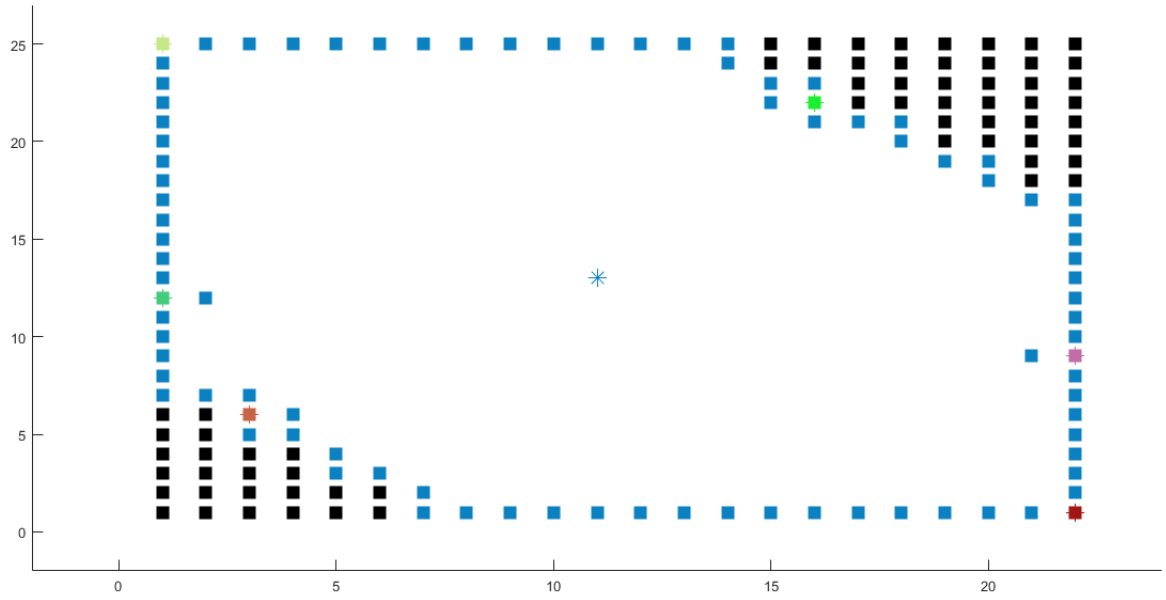


Figure 5.1 Initial conditions for simulation 1

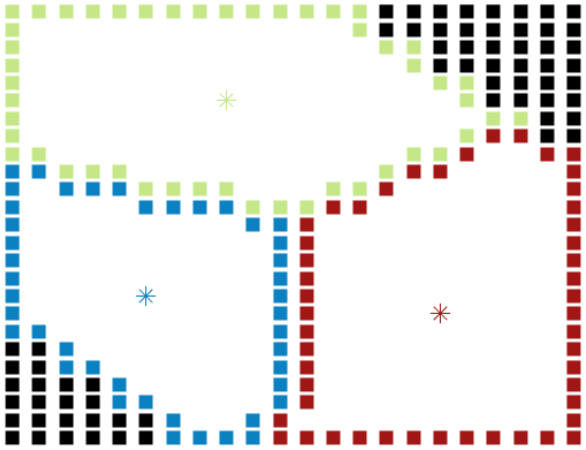


Figure 5.2 Final partition with 3 robots

Figure 5.3 Final partition with 4 robots

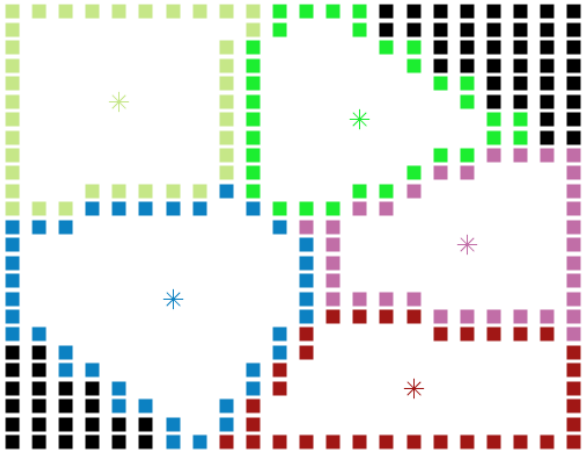


Figure 5.4 Final partition with 5 robots

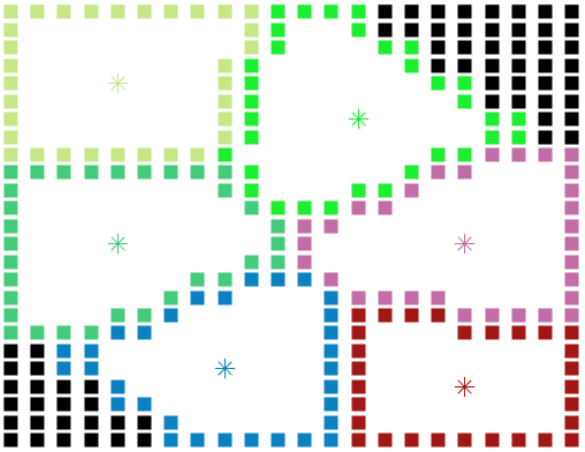


Figure 5.5 Final partition with 6 robots

Figure 5.6 Final partition with 7 robots

5.1.1. Results using Synchronous Gossiping coverage algorithm

The results are shown in Table 5.1 the value of the cost functional and the number of iterations that have been done in order to achieve the final partition. At each iteration each robot communicates.

Number of robots	**3**	**4**	**5**	**6**	**7**
Cost functional	2469.0	2085.7	1886.3	1699.9	1576.8
Time	10	13	14	20	20
Amount of Communications	30	52	70	120	140

Table 5.1 Results of simulation 1 using Synchronous Gossiping coverage algorithm

The evolution of the costs functional is shown in Figure 5.7. As it can be observed the number of robots reduces $H_{multicenter}$, which has been written as H to abbreviate. Another fact of interest is that as the number of robots increases the speed of minimization decreases, due to the increase in the number of communications between the robots.

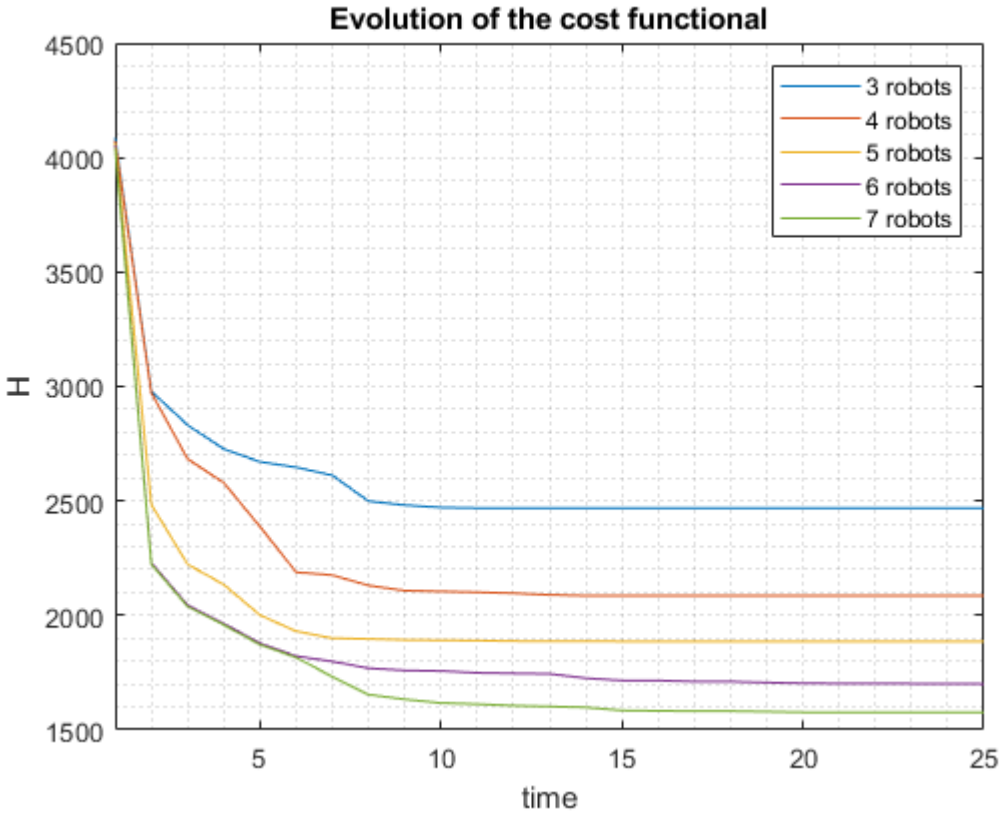


Figure 5.7 Evolution of the cost functional for different number of robots in simulation 1

5.1.2. Results using Asynchronous Broadcast coverage algorithm

The main difference in the Asynchronous Broadcast is the amount of iterations that requires achieving the final partition. To understand that, the study of the algorithm is done with 100 simulations. In the next table, there are the media and variance of the amount of iterations and the amount of communications for each team of robots:

Amount of Communications					
Number of robots	3	4	5	6	7
Mean (μ)	32.39	42.90	60.61	71.87	87.41
Variance (σ)	5.9664	12.3187	25.2154	27.3381	37.0907

Table 5.2 Amount of communications using Asynchronous Broadcast coverage algorithm

Time					
Number of robots	3	4	5	6	7
Mean (μ)	98.91	119.62	152.99	175.02	220.94
Variance (σ)	19.2193	38.1664	56.8413	61.6944	91.0859

Table 5.3 Time using the Asynchronous Broadcast coverage algorithm

The results show that, if the number of robots increases, the number of times that happens a communication increase.

5.1.3. Comparative analysis

As it can be observed, the main difference in both algorithms resides in the fact that the second is asynchronous and that at each iteration only communicates one robot instead of each one of them, which increases the amount of iterations to be able to achieve the final partition. However, the amount of communications is reduced due to the broadcast protocol communication. Which would reduce the energy consumption for the robots and the data redundancy. In Table 5.4, Figure 5.8 , it can be observed the differences explained in this paragraph.

Partition	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Percentage of reduction in the number of communications	65.90%	98.80%	72.15%	51.74%	58.98%	73.94%	70.25%

Table 5.4 Percentage of reduction in the number of communications between both algorithms

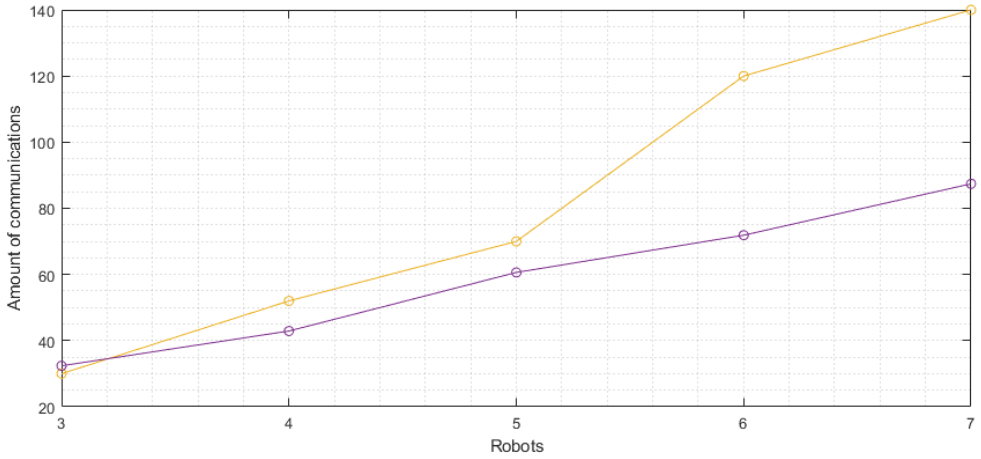


Figure 5.8 Comparison of the amount of communications between the Synchronous Gossiping coverage algorithm and the Asynchronous Broadcast coverage algorithm. The purple one is using the Asynchronous Broadcast coverage algorithm and the yellow one is using the Synchronous Gossiping coverage algorithm

5.2. Simulation 2

This simulation consists in simulating a group of 5 robots in a grid map of 20x24, with different size of obstacles. The initial conditions have been chosen randomly. Figure 5.9 to the first partition of the map with 5 robots. In this case, the simulation is always with 5 robots, but their initial conditions will be different in order to see how time and the amount of communications changes due to the initial conditions, and how the different partitions that can be achieved can make the final value for the cost functional differ between them. Thanks to this kind of simulations, the user knows the best way to initialize the system. The initial partitions and final partitions for simulation 2 with the Synchronous Gossiping coverage algorithm are:

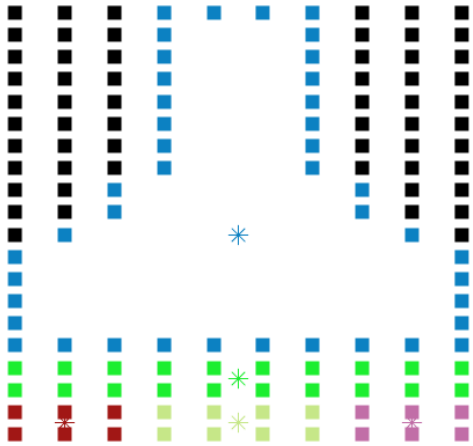


Figure 5.9 Simulation 2, initial partition 1

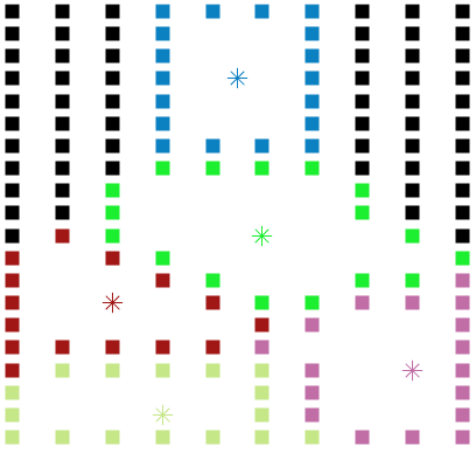


Figure 5.10 Simulation 2, final partition 1

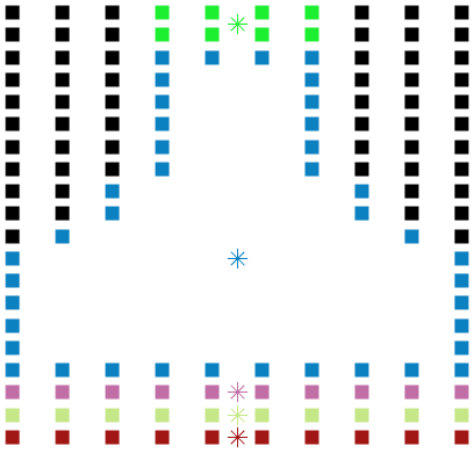


Figure 5.11 Simulation 2, initial partition 2

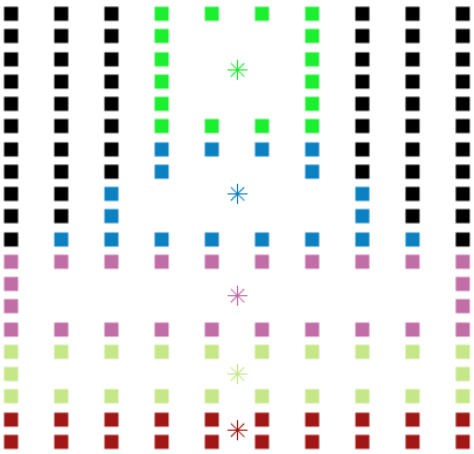


Figure 5.12 Simulation 2, final partition 2

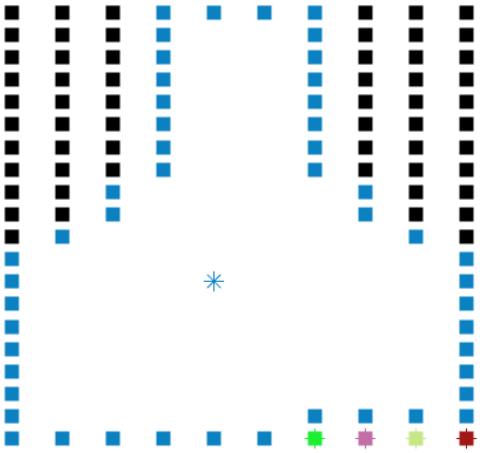


Figure 5.13 Simulation 2, initial partition 3

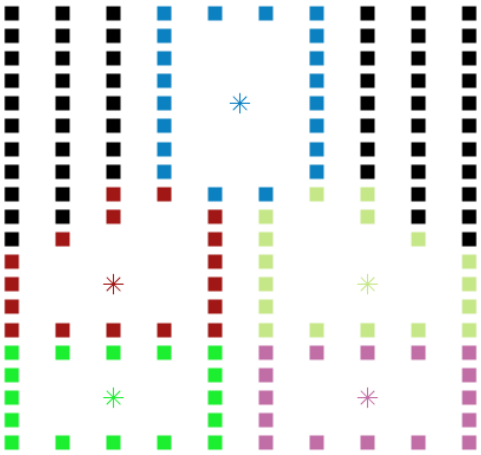


Figure 5.14 Simulation 2, final partition 3

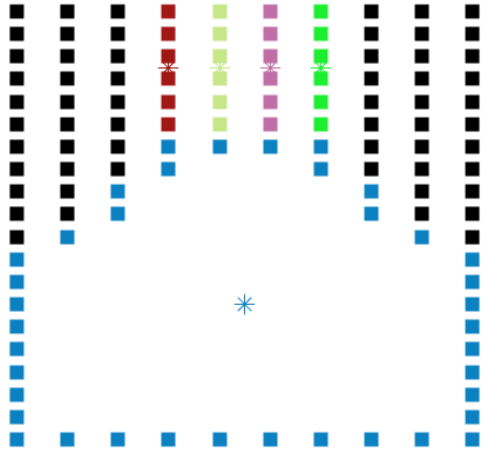


Figure 5.15 Simulation 2, initial partition 4

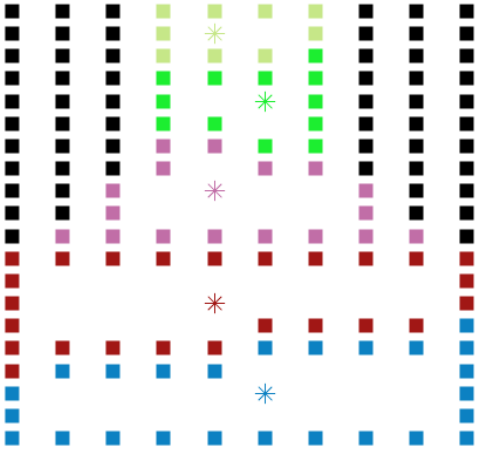


Figure 5.16 Simulation 2, final partition 4

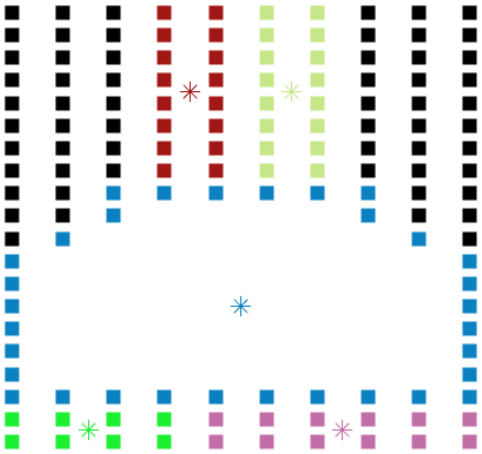


Figure 5.17 Simulation 2, initial partition 5

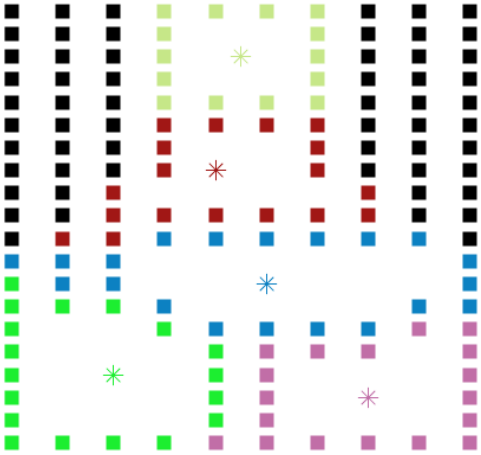


Figure 5.18 Simulation 2, final partition 5

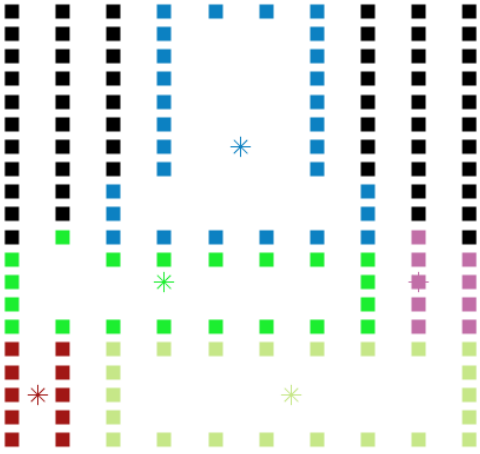


Figure 5.19 Simulation 2, initial partition 6

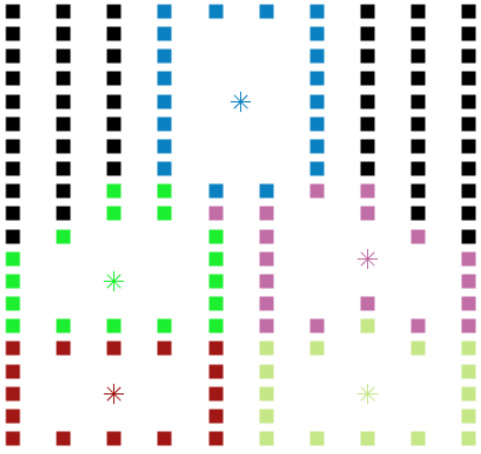


Figure 5.20 Simulation 2, final partition 6

5.2.1. Results using Synchronous Gossiping coverage algorithm

It is interesting in this case using the gossiping coverage algorithm to study how the cost functionals have developed and to study how many iterations the algorithm has needed to do it. The results are shown in the next table:

Partition	**Start Cost Functional**	**Final Cost Functional**	**Time**	**Number of communications**
1	533.7847	303.6634	16	80
2	530.0781	348.9404	10	50
3	718.2359	301.7413	16	80
4	536.6069	350.3765	22	110
5	419.6988	305.3647	16	80
6	367.3831	302.0762	10	50

Table 5.5 Results of the simulation 2

As it can be observed, a main consequence of changing the initial conditions is the different values for the cost functional, from the first iteration until the last one.

In Figure 5.21 can be observed the evolution of the cost functionals for the 6 different initial partitions.

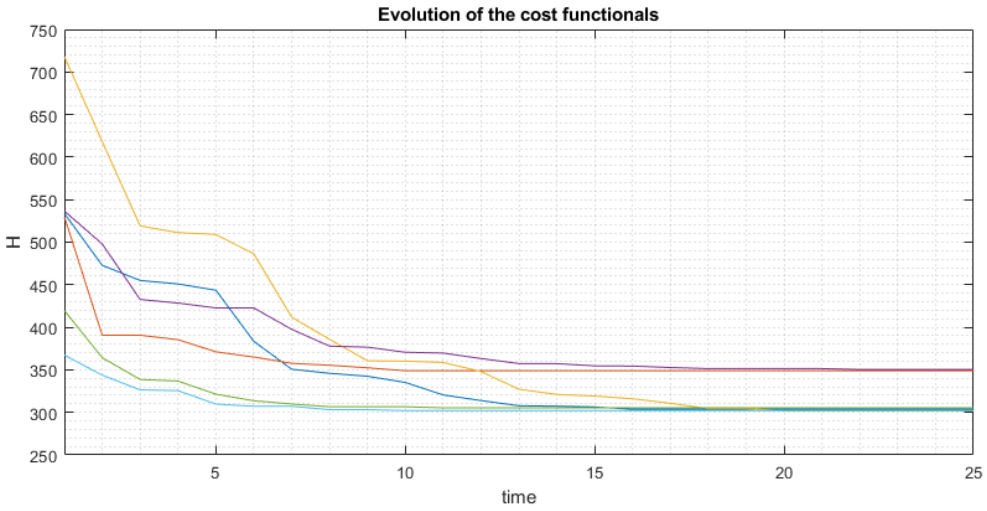


Figure 5.21 Evolution of the cost functional for each partition in simulation 2

5.2.2. Results using Asynchronous Broadcast coverage algorithm

To be able to understand and take conclusions for the different results achieved by the Asynchronous Broadcast coverage algorithm, each partition is going to be simulated 100 times as in simulation 1.

Amount of communications						
Partition	1	2	3	4	5	6
Mean (μ)	52.72	49.40	57.72	56.91	47.19	36.97
Variance (σ)	11.4395	13.0023	22.0400	18.5871	28.0145	20.3034
Time						
Partition	1	2	3	4	5	6
Mean (μ)	138.40	127.80	153.35	150.30	124.04	99.55
Variance (σ)	31.8236	36.5604	57.9830	55.1479	76.3116	54.0004

Table 5.6 Results using Asynchronous Broadcast coverage algorithm for 100 simulations

5.2.3. Comparative analysis

As it happened in simulation 1, the results using the Asynchronous Broadcast coverage algorithm give us a slower convergence to the final partition, but with less amount of communications. The results using the Asynchronous Broadcast reduced in a significant way the amount of communications. In is shown the percentage of reduction of the amount of communications using the Asynchronous Broadcast and the Synchronous Gossiping.

Partition	**1**	**2**	**3**	**4**	**5**	**6**	**Average**
Percentage	65.90%	98.80%	72.15%	51.74%	58.98%	73.94%	70.25%

Table 5.7 Percentage of reduction in the amount of communications between both algorithms

5.3. Simulation 3

In this subchapter it is going to be studied a grid map of 15x15 with teams of 4, 6 and 8 robots and there will be 3 different initial conditions for each team robots, which will permit to take all the necessary conclusions about the differences of using both algorithms. The map is shown in Figure 5.22.

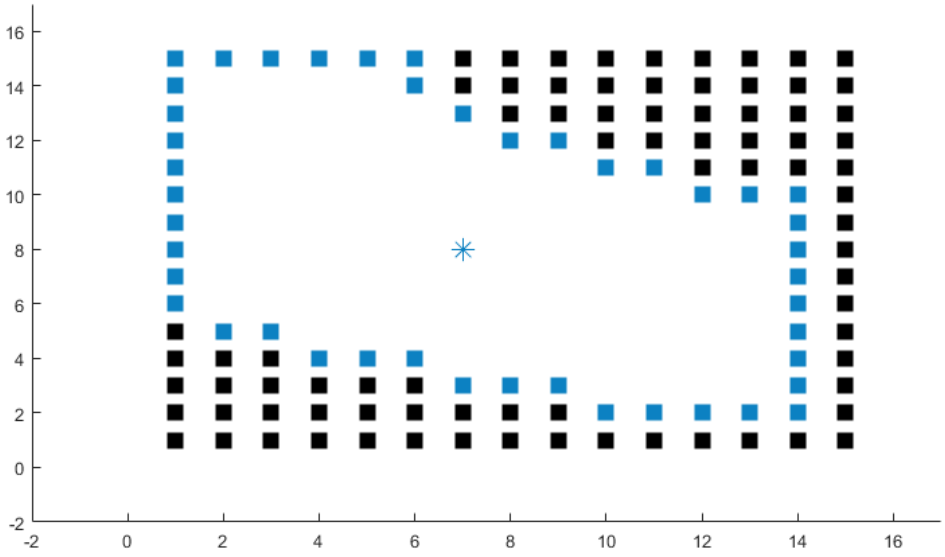


Figure 5.22 Discretized environment in a 15x15 grid map for simulation 3

5.3.1. Simulation 3 with 4 robots

The initial partitions and the evolution of the cost functional using the Synchronous Gossiping coverage algorithm for the team of four robots are:

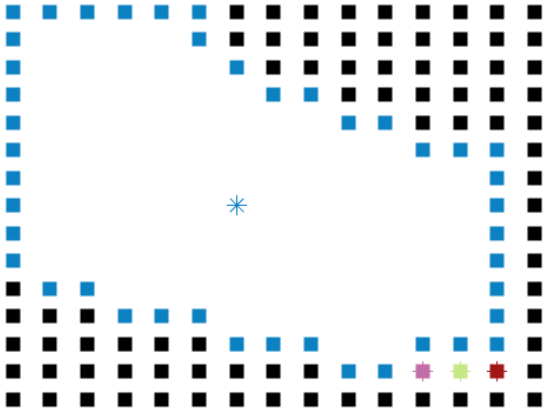


Figure 5.23 Simulation 3 with 4 robots, initial partition 1

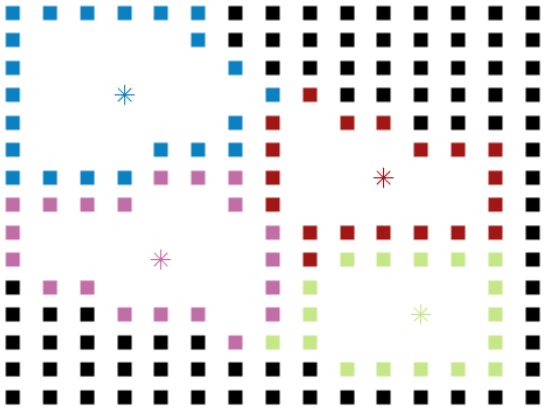


Figure 5.24 Simulation 3 with 4 robots, final partition 1

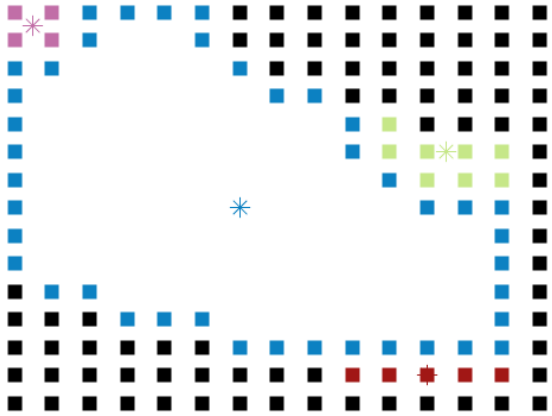


Figure 5.25 Simulation 3 with 4 robots, initial partition 2

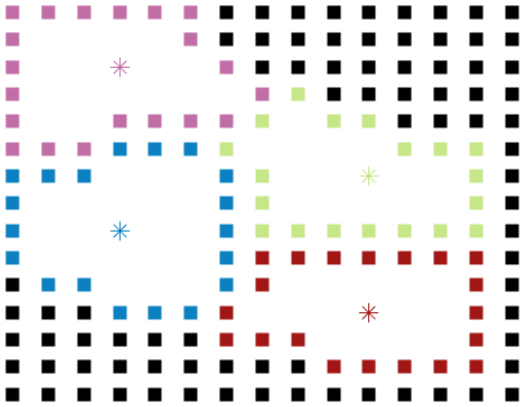


Figure 5.26 Simulation 3 with 4 robots, final partition 2

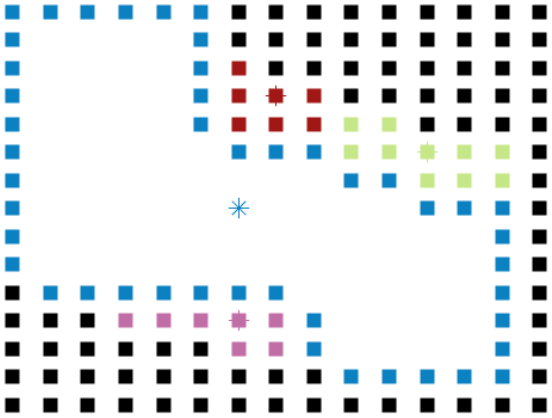


Figure 5.27 Simulation 3 with 4 robots, initial partition 3

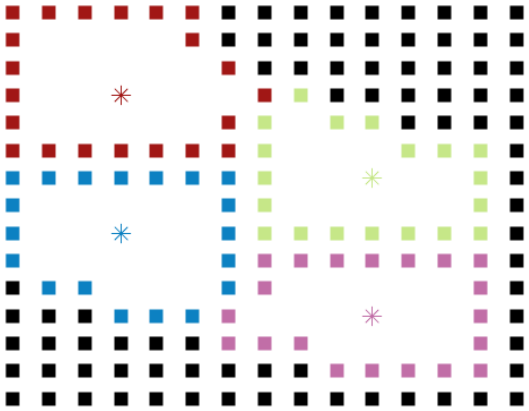


Figure 5.28 Simulation 3 with 4 robots, final partition 3

As it has been shown, the final partitions are almost the same, however the evolution of the cost functional and their final values are different. This can be seen in Figure 5.29.

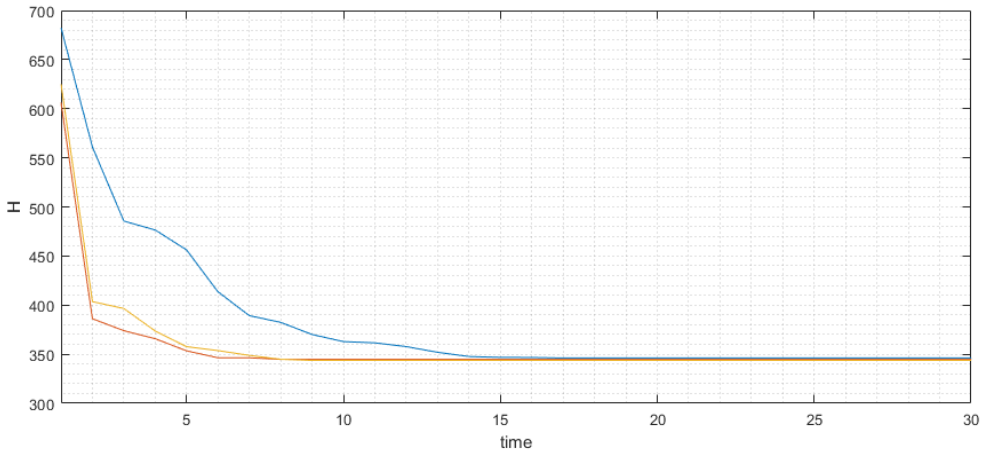


Figure 5.29 Evolution of the cost functional for each partition. The blue one is the 1st partition, the red one is 2nd partition and the yellow one is the 3rd partition.

Partition	Start Cost Functional	Final Cost Functional	Time	Number of communications
1	682.3261	346.1458	15	60
2	606.2446	344.7653	6	24
3	625.0551	343.6813	9	36

Table 5.8 Results using the Synchronous Gossiping coverage algorithm

5.3.2. Results using the Asynchronous Broadcast coverage algorithm with 4 robots

Each partition has been simulated 100 times. In the next table can be find the amount of communications and the time that it takes to achieve the final partition for each case.

Amount of communications			
Partition	1	2	3
Mean (μ)	46.22	49.06	50.44
Time			
Partition	1	2	3
Mean(μ)	124.56	137.60	138.96

Table 5.9 Results using the Asynchronous Broadcast coverage algorithm

5.3.3. Simulation 3 with 6 robots

The initial partitions and the evolution of the cost functional using the Synchronous Gossiping coverage algorithm for the team of six robots are:

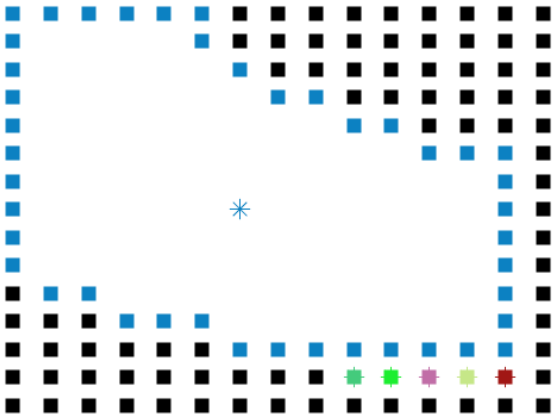


Figure 5.30 Simulation 3 with 6 robots, initial partition 1

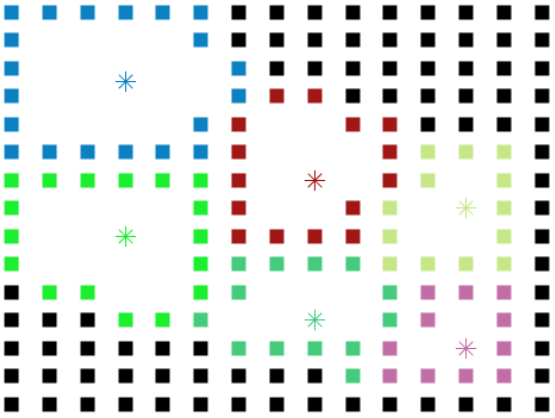


Figure 5.31 Simulation 3 with 6 robots, final partition 1

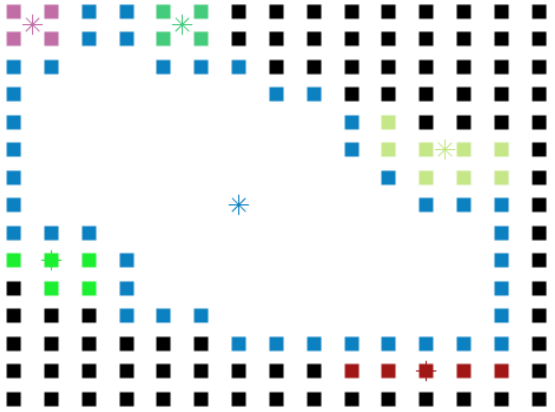


Figure 5.32 Simulation 3 with 6 robots, initial partition 2

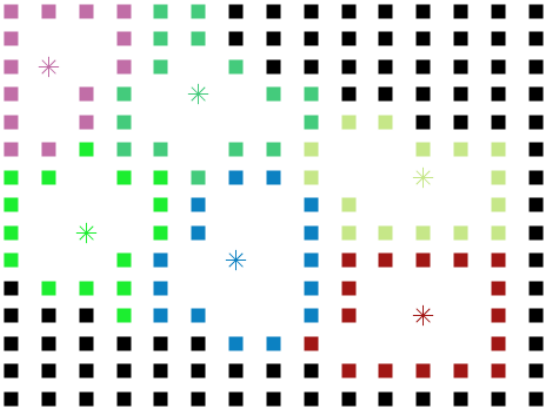


Figure 5.33 Simulation 3 with 6 robots, final partition 2

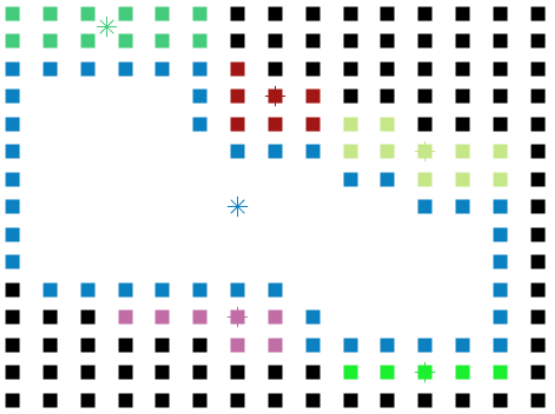


Figure 5.34 Simulation 3 with 6 robots, initial partition 3

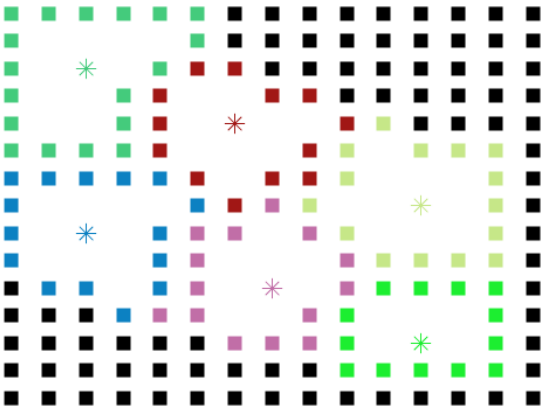


Figure 5.35 Simulation 3 with 6 robots, final partition 3

The evolution of the cost functional for the three partitions is:

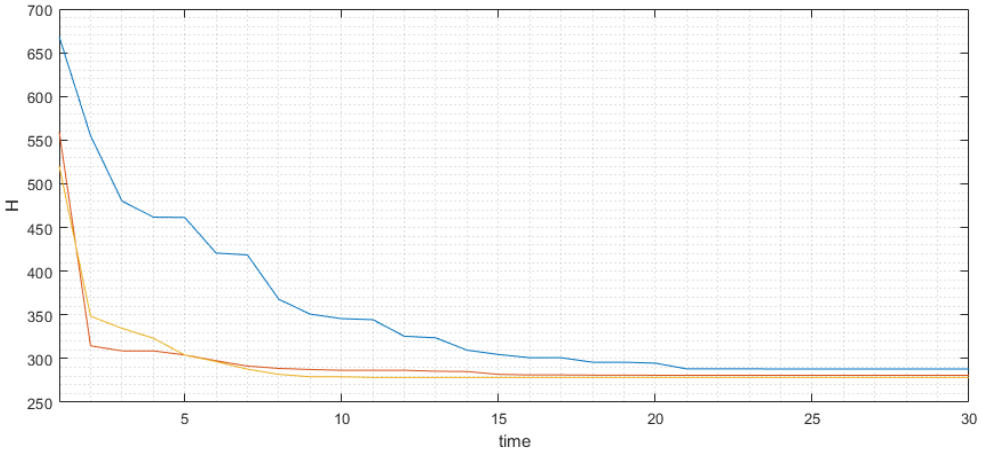


Figure 5.36 Evolution of the cost functional for each partition. The blue one is the 1st partition, the red one is 2nd partition and the yellow one is the 3rd partition.

Partition	**Start Cost Functional**	**Final Cost Functional**	**Time**	**Number of communications**
1	668.4068	288.0308	21	126
2	559.7159	280.8724	15	90
3	520.4985	278.3802	9	54

Table 5.10 Results using the Synchronous Gossiping coverage algorithm

5.3.4. Results using the Asynchronous Broadcast coverage algorithm with 6 robots

Each partition has been simulated 100 times. In the next table can be find the amount of communications and the time that it takes to achieve the final partition for each case.

Amount of communications			
Partition	1	2	3
Mean (μ)	65.72	62.82	48.9
Time			
Partition	1	2	3
Mean(μ)	169.08	162.18	133.74

Table 5.11 Results using the Asynchronous Broadcast coverage algorithm

5.3.5. Simulation 3 with 8 robots

The initial partitions and the evolution of the cost functional using the Synchronous Gossiping coverage algorithm for the team of eight robots are:

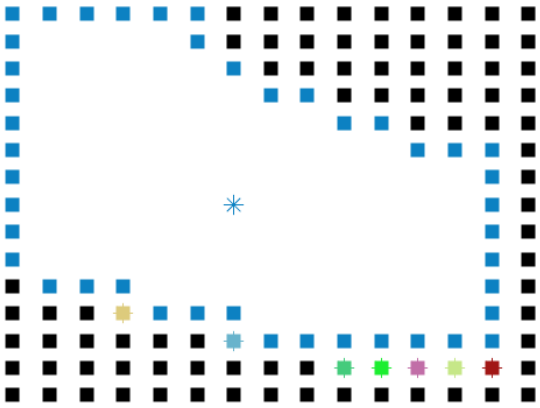


Figure 5.37 Simulation 3 with 8 robots, initial partition 1

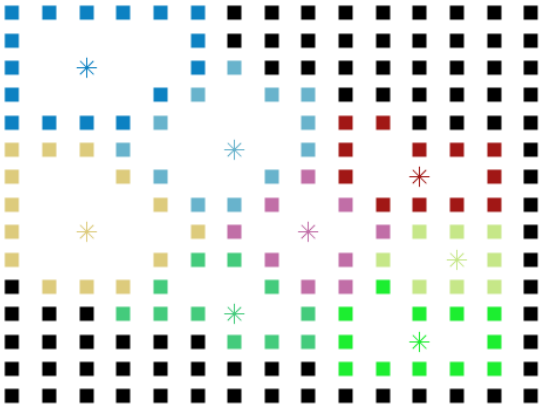


Figure 5.38 Simulation 3 with 8 robots, final partition 1

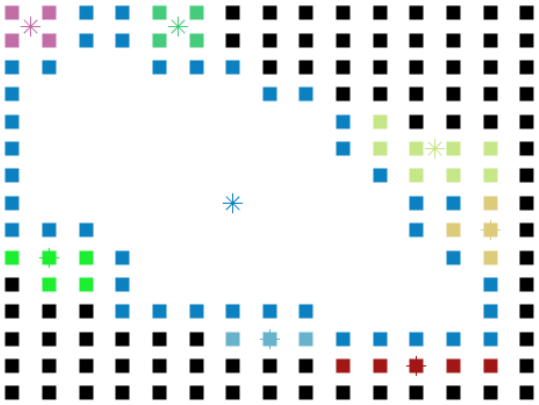


Figure 5.39 Simulation 3 with 8 robots, initial partition 2

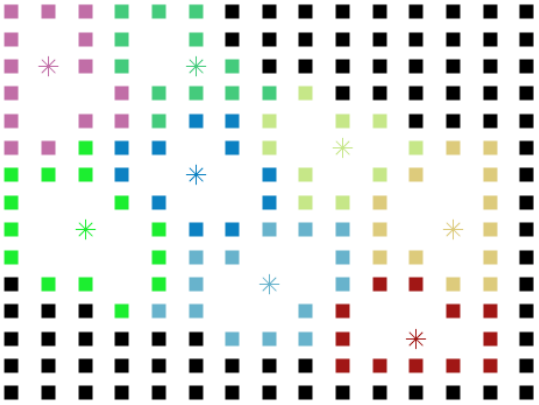


Figure 5.40 Simulation 3 with 8 robots, final partition 2

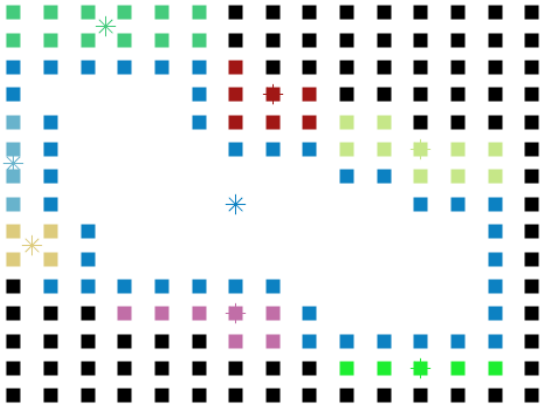


Figure 5.41 Simulation 3 with 8 robots, initial partition 3

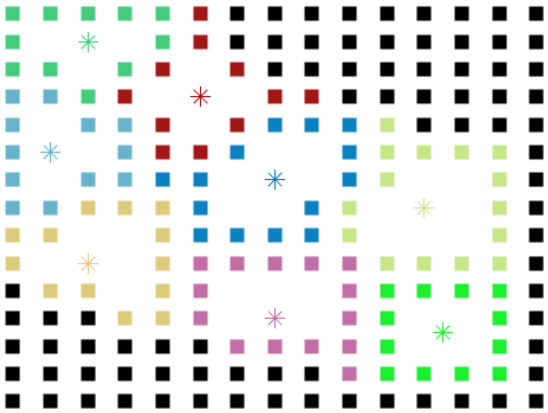


Figure 5.42 Simulation 3 with 8 robots, final partition 3

The evolution for the cost functional of the three partitions is:

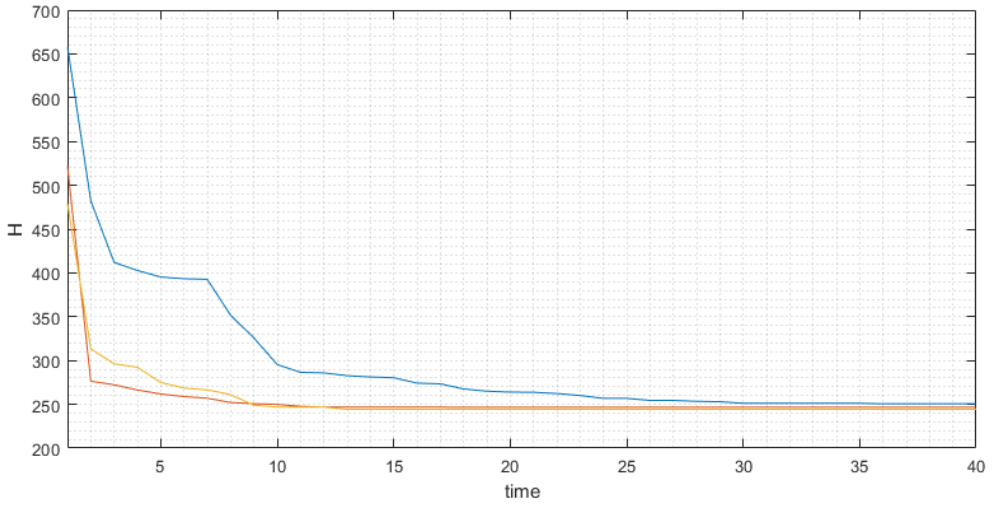


Figure 5.43 Evolution of the cost functional for each partition. The blue one is the 1st partition, the red one is 2nd partition and the yellow one is the 3rd partition.

Partition	**Start Cost Functional**	**Final Cost Functional**	**Time**	**Number of communications**
1	658.4068	250.6092	36	288
2	521.7978	246.6024	18	144
3	479.3199	244.2610	12	96

Table 5.12 Results using the Synchronous Gossiping coverage algorithm

5.3.6. Results using the Asynchronous Broadcast coverage algorithm with 8 robots

Each partition has been simulated 100 times. In the next table can be find the amount of communications and the time that it takes to achieve the final partition for each case.

Amount of communications			
Partition	1	2	3
Mean (μ)	93.51	73.5	72.08
Time			
Partition	1	2	3
Mean(μ)	234.56	186.96	180.88

Table 5.13 Results using the Asynchronous Broadcast coverage algorithm

5.3.7. Comparative analysis

The results in simulation 3 show that the Asynchronous Broadcast coverage algorithm is very useful for reducing the amount of communications in big teams of robots compared to the Synchronous Gossiping coverage algorithm. However, in small teams of robots, where there are less communications due to the number of robots, as in the team of 4 robots the number of communications is lower or similar to the Synchronous Gossiping coverage algorithm.

Moreover, it has been proved another time how important the decision of choosing good initial partitions is for each robot of the team. The results from partition number 1 and partition number 3, show a big contrast in the amount of communications.

In the next is shown how the difference between both algorithms increases each time that the number of robots increase for each partition:

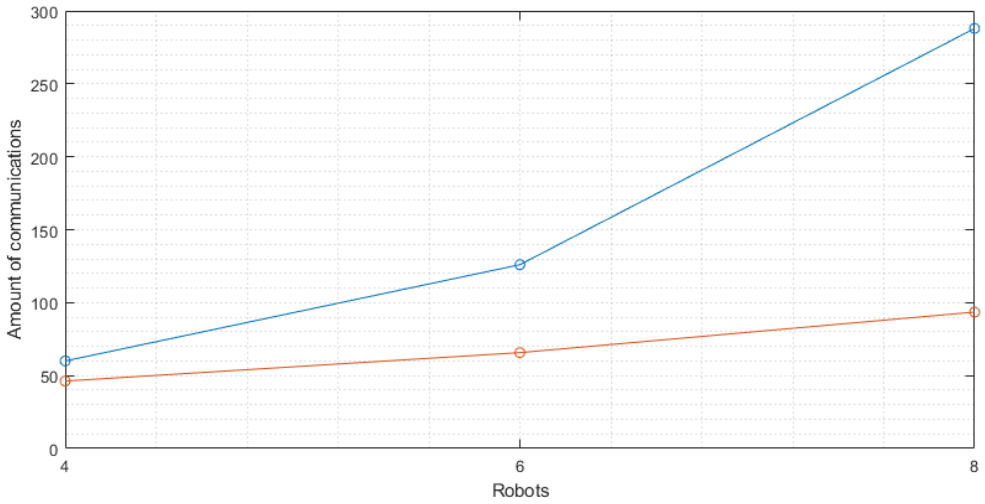


Figure 5.44 Amount of communications for each team robots in partition 1. The blue one is using the Synchronous Gossiping coverage algorithm and the red one is using the Asynchronous Broadcast coverage algorithm

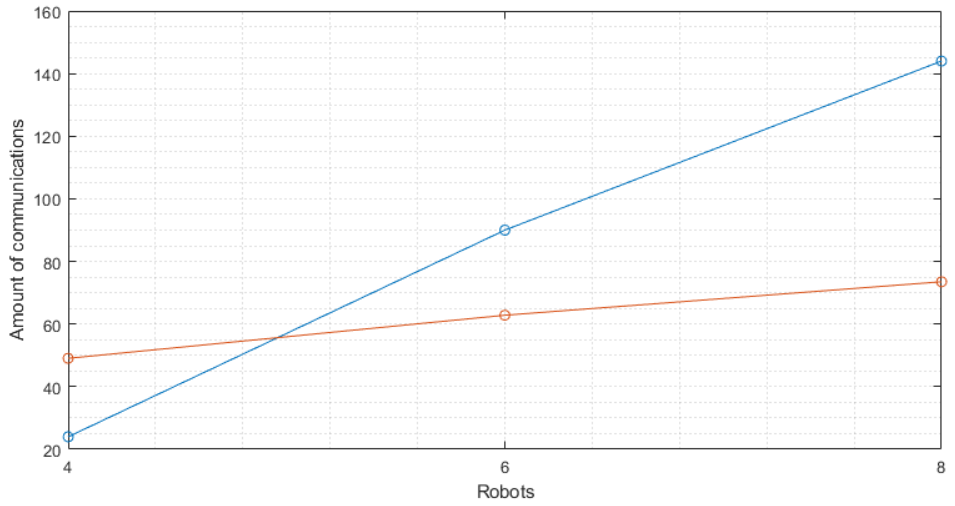


Figure 5.45 Amount of communications for each team robots in partition 2. The blue one is using the Synchronous Gossiping coverage algorithm and the red one is using the Asynchronous Broadcast coverage algorithm

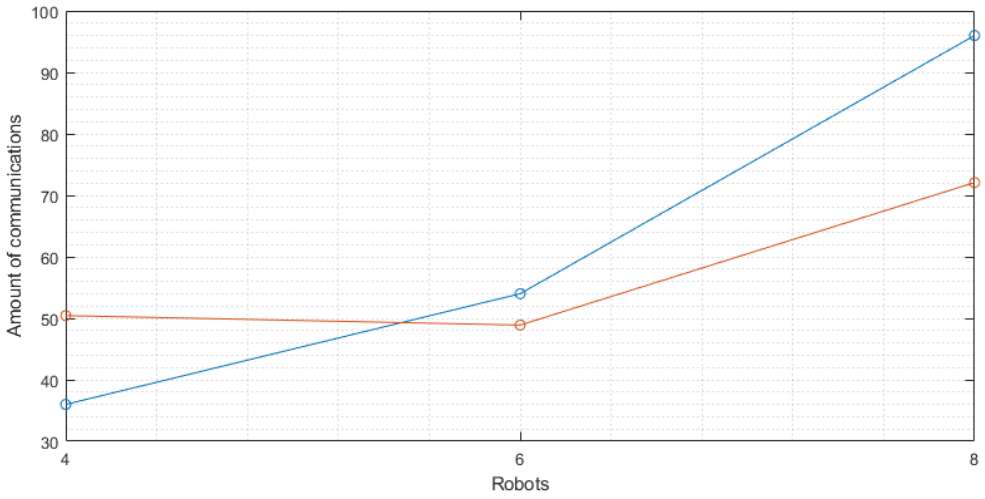
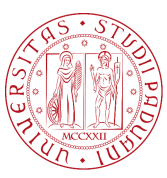


Figure 5.46 Amount of communications for each team robots in partition 3. The blue one is using the Synchronous Gossiping coverage algorithm and the red one is using the Asynchronous Broadcast coverage algorithm



6. Conclusions and Future work

In this thesis has been presented how to deal with distributed partitioning and coverage control problems for a network of robotic agents in a non-convex environment.

It has been formally described how to discretize an environment as a connected graph and has been discussed important properties of the setup, such as that the centroid of a robot's region always belongs to the region, or that each robot's region remains connected during the evolution of the algorithm.

Moreover, it has been described how two coverage algorithms on a graph guarantees convergence to a centroidal Voronoi partition using only pairwise communication. What's more, it has been presented an implemented the Asynchronous Broadcast coverage algorithm for discretized environments and for large groups of robots. Moreover, it has been proved that by using the Asynchronous Broadcast coverage algorithm the communications requirements are reduced. In fact, the amount of communications is more reduced in a significant way as the teams of robots and the maps are larger.

The main vision after working on that project is that partitioning and coverage algorithms will form the foundation of a distributed task servicing setup for teams of mobile robots. The main idea is that the robots would split their time between servicing tasks in their territory and moving to contact to their neighbors and improve the coverage of the space. Some of these tasks are well known as search and rescue, surveillance of public space or service respond in an industrial or farm factory.

On the basis of this work, future projects can be developed.

Firs of all, some implementations in the code can be added by putting weight on the edges and simulate the new partitions achieved and how the teams of robots converge and by developing new ways to compute the centroids and the distance matrices.

Finally, to perform hardware experiments of the algorithm which will require a motion protocol for the robot to guarantee that they make contact with their neighbors. And simulate the teams of robots in different environments using both communication protocols.

7. Bibliography

[1] J. W. Durham, R. Carli, P. Frasca and F. Bullo, "Discrete Partitioning and Coverage Control for Gossiping Robots," *IEEE Transaction on Robotics*, vol. 28, no. 2, pp. 364-378, April 2012.

[2] J. W. Durham, R. Carli, P. Frasca and F. Bullo, "Discrete Partitioning and Coverage Control with gossip communication," January 2009.

[3] J. W. Durham, R. Carli and F. Bullo, "Pairwise optimal coverage control for robotic networks in discretized environments," in *IEEE Conf. on Decision and Control*, Atlanta, GA, USA, December 2010.

[4] J. Cortés, S. Martínez, T. Karatas and F. Bullor, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243-255, April 2004.

[5] N. Bof, R. Carli, A. Cenedese and L. Schenato, "Asynchronous Distributed Camera Network Patrolling under Unreliable Communication," *IEEE Transactions on Automatic Control*, vol. 62, no. 11, pp. 5982-5989, Nov. 2017.

Appendix

Main Functions for the Synchronous Gossip coverage algorithm

Main function

```
function [M,H_list,ciclos] = SyncGossip(M,n)
%Initialize the different variables
count_loop = 0;
flag_loop = false;
H_list = [];
count_H = 1;
ciclos =0;
[f,c]=size(M);
%Calculate subgraph matrix M of each robot(cells occupied by each robot)
MM = redefineN(M,n);
%Calculate the center of each submatrix
positionN=optimalN(M,MM,n);
%Calculate the ocupation Matrix of each robot and convert the matrix to 0-1
%matrices
ON = ocupationN(M,n);
KN = letspaintN(ON,M,n);
%Generates a vector of N rows with the cells that have each robot as
%frontier
[xN,yN] = drawingN(KN,n);
%generate a vector of colors for each robot
colors = zeros(n,3);
for i =1:n
    R = rand();
    G = rand();
    B = rand();
    colors(i,1) = R;
    colors(i,2) = G;
    colors(i,3) = B;
end
%Plot the initial conditions
picaso(xN,yN,positionN,n,M,colors)
pause(2)
%Find the Distance total matrix and each distance subset matrix
[~,DD] = distance_matrixN2(f,c,MM,n,positionN);
%Calculate the H for each region
H_old = calculusH(DD);
H=H_old;
%initialize the list of cost functionals
H_list(count_H) = H;
count_H = count_H + 1;
while flag_loop == false
    %Starts the loop
    for i =1:n
        ciclos = ciclos+1;
        [M,H,positionN] = Main_SGC(f,c,M,i,positionN,n);
        ON = ocupationN(M,n);
        KN = letspaintN(ON,M,n);
        [xN,yN] = drawingN(KN,n);
        %Plot after one robot communicates with the others
        picaso(xN,yN,positionN,n,M,colors)
        H_list(count_H) = H;
        count_H = count_H + 1;
        if length(H_list) >2
```

```
            for k = 2:length(H_list)
                %To see if the value of H is being reapeted which means
                %that the convergence is achieved
                if H_list(k) ==H_list(k-1)
                    count_loop = count_loop+1;
                else
                    count_loop = 0;
                end
            end
        end
        %When this condition is achieved, it stops de loop
        if count_loop > 26
            flag_loop = true;
        end
    end
end
%Calculate tha occupancy matrices and creates the vector of positions
ON = ocupationN(M,n);
KN = letspaintN(ON,M,n);
[xN,yN] = drawingN(KN,n);
%Plot the last drawn
picaso(xN,yN,positionN,n,M,colors)
pause(0.7)
end
```

Pairwise function

```
function [M, H,positionN] = Main_SGC(f,c,M,number,positionN,n)
%Obtain the occupancy matrix for each robot
MM = redefineN(M,n);
%Find the Distance total matrix and each distance subset matrix
[D,DD] = distance_matrixN2(f,c,MM,n,positionN);
%Calculate the H for each region
H_old = calculusH(DD);
%Create a Matrix to see the neighbors of each robot
C = contactN(M,n);
%Begin a loop
for j =1:n
    %If they are neighbors they exchange territories
    if C(number,j) ==1
        %Find their distance matrices
        DD1 = DD(1+f*(number-1):number*f,:);
        D1 = D(1+f*(number-1):number*f,:);
        DD2 = DD(1+f*(j-1):j*f,:);
        D2 = D(1+f*(j-1):j*f,:);
        %Compute the new distribution
        M_new= DistributionN(M,number,j,D1,D2,DD1,DD2);
        %Compute new occupancy matrices
        [MM_new] = redefineN(M_new,n);
        %Compute the new positions
        positionN_new = optimalN2(f,c,MM_new,n);
        MM1_new=MM_new(1+f*(number-1):number*f,:);
        MM2_new=MM_new(1+f*(j-1):j*f,:);
        %Eliminite the points that are not still connected by a path to
        %the center of the robot
        MMM_1 = recreation(MM1_new,number,positionN_new(number,:));
        MMM_2 = recreation(MM2_new,j,positionN_new(j,:));
        for k =1:f
            for h =1:c
                if MM1_new(k,h)~=MMM_1(k,h)
                    MM2_new(k,h)=j;
                end
```

```
                if MM2_new(k,h)~=MMM_2(k,h)
                    MM1_new(k,h)=number;
                end
            end
        end
        %New values of distance
        [D_new,DD_new] = distance_matrixN2(f,c,MM_new,n,positionN_new);
        %New cost functional
        H_new=calculusH(DD_new);
        %If its lower, then update
        if H_new <H_old
            M=M_new;
            MM=MM_new;
            positionN=positionN_new;
            D=D_new;
            DD=DD_new;
            C=contactN(M,n);
        end
    end
end
%New distances
[~,DD] = distance_matrixN2(f,c,MM,n,positionN);
%Calculate the H for each region
H = calculusH(DD);
end
```

Main Functions for the Asynchronous Broadcast coverage algorithm

Main function

```
function [M_gen,H_list,ciclos] = AsyncBroadcast3(M,n)
%Initialize the different variables
count_loop = 0;
flag_loop = false;
H_list = [];
count_H = 1;
ciclos =0;
[f,c]=size(M);
%Generate a Matrix per each robot
M_gen=zeros(f*n,c);
%Calculate subgraph matrix M of each robot(cells occupied by each robot)
MM = redefineN(M,n);
%Calculate the center of each submatrix
positionN=optimalN(M,MM,n);
%Calculate the ocupation Matrix of each robot and convert the matrix to 0-1
%matrices
ON = ocupationN(M,n);
KN = letspaintN(ON,M,n);
%Generates a vector of N rows with the cells that have each robot as
%frontier
[xN,yN] = drawingN(KN,n);
%generate a vector of colors for each robot
colors = zeros(n,3);
for i =1:n
    M_gen(1+f*(i-1):i*f,:)=M;
    R = rand();
    G = rand();
    B = rand();
    colors(i,1) = R;
    colors(i,2) = G;
```

```
    colors(i,3) = B;
end
%Generate the vector of obstacles
obs=[];
for i =1:f
    for j =1:c
        if M(i,j)==1000
        obs =vertcat(obs,[i,j]);
        end
    end
end
%Plots the initial conditions
picaso(xN,yN,positionN,n,M,colors)
pause(2)
%Find the Distance total matrix and each distance subset matrix
[~,DD] = distance_matrixN2(f,c,MM,n,positionN);
%Calculate the H for each region
H_old = calculusH(DD);
H = H_old;
repeat=0;
while flag_loop == false
    for i =1:n
        %Generates a random number
        Random = rand();
        %If its lower than 0.4, the robot can communicate with its
        %neighbors
        if Random < 0.4
            if i~=repeat
                repeat=i;
                ciclos = ciclos+1;
                [M_gen,H,positionN] =
Main_ABC3(f,c,M_gen,positionN,i,n,colors,obs);
            end
        end
        %Update H
        H_list(count_H) = H;
        count_H = count_H + 1;
    end
    if length(H_list) >2
        for k = 2:length(H_list)
            if H_list(k-1)==H_list(k)
                count_loop = count_loop+1;
            else
                count_loop = 0;
            end
        end
    end
    %Condition to see if the system can not be minimized
    if count_loop >25
        flag_loop = true;
    end
end
MM = redefineN(M_gen,n);
%Calculate the occupancy matrices and the vector of positions
ON = ocupationN(M_gen(1:f,1:c),n);
KN = letspaintN(ON,M_gen(1:f,1:c),n);
[xN,yN] = drawingN(KN,n);
%Plot the final partition
picaso_proces(xN,yN,positionN,n,M_gen(1:f,1:c),colors,n,j)
pause(0.7)
end
```

Pairwise function

```
function [M_gen, H,positionN] =
Main_ABC3(f,c,M_gen,positionN,number,n,colors,obs)
%Initialize variables
M_fake=zeros(f,c);
flag_p=false;
flag_d=false;
Lucho =zeros(f*n,c);
%obtain the matrix MM of the robot that sends data and create support
%variables
MMs=redefineN(M_gen(1+f*(number-1):number*f,:),n);
Lucho(1+f*(number-1):number*f,:)=MMs(1+f*(number-1):number*f,:);
MM_1 = MMs(1+f*(number-1):number*f,:);
MO=MM_1;
P_tot =positionN;
%Find the Distance total matrix and each distance subset matrix
[D,DD] = distance_matrixN2(f,c,MMs,n,positionN);
DD_1 = DD(1+f*(number-1):number*f,:);
D_1 = D(1+f*(number-1):number*f,:);
%Calculate the H for each region
H_old = calculusH(DD);
H = H_old;
%Create the matrix of neighbors
C = contactN(M_gen(1+f*(number-1):number*f,:),n);
list=[];
list_2 = [];
%Create a Matrix MM of support of the robot that sends data
MM_num=redefineN(M_gen(1+f*(number-1):number*f,:),n);
%Create a infinite loop
for j =1:n
    %Finds matrix MM of robot j
    MM=redefineN(M_gen(1+f*(j-1):j*f,:),n);
    MM_1=MO;
    %Sends data to the robot j, to update their state in case that
    %they have communicated data before
    if j~=number
        Lucho(1+f*(j-1):j*f,:)= MM(1+f*(j-1):j*f,:);
        for q=1:f
            for t=1:c
                if MM_num(q+f*(number-1),t)==number
                    M_gen(q+f*(j-1),t)=MM_num(q+f*(number-1),t);
                end
            end
        end
    end
    %If robot J is neighbor
        if C(number,j) ==1
            flag_d=true;
            %Create variables to support the proces
            MM_gen=redefineN(M_gen(1+f*(j-1):j*f,:),n);
            %Calculate its distance matrices
            [D,DD] = distance_matrixN2(f,c,MM_gen,n,positionN);
            DD_1 = DD(1+f*(number-1):number*f,:);
            D_1 = D(1+f*(number-1):number*f,:);
            %Create support variables for each robot MM
            MM_new = MM_gen;
            MM_2_old=MM(1+f*(j-1):j*f,:);
            MM_1 =MM_new(1+f*(number-1):number*f,:);
            MM_2 = MM_new(1+f*(j-1):j*f,:);
            %Search the support matrices of distance
            [D_1,D2_support] = SearchDN2(D,number,j,n,f);
            [DD_1,DD2_support] = SearchDN2(DD,number,j,n,f);
```

```
            %Compute New distribution
            [MM1_new,MM2_new] =
DistributionN2(f,c,MM_1,MM_2,number,j,D_1,D2_support,DD_1,DD2_support);
            %Recompute MM and center point
            MM_new(1+f*(j-1):j*f,:) = MM2_new;
            MM_new(1+f*(number-1):number*f,:) = MM1_new;
            positionN_new = optimalN2(f,c,MM_new,n);
            P_tot(j,1)=positionN_new(j,1);
            P_tot(j,2)=positionN_new(j,2);
            PN = P_tot(number,:);
            PN2 =P_tot(j,:);
            %Eliminate points that hasn't a path to the center of the robot
            MM_1 = recreation(MM1_new,number,PN);
            MM_2 = recreation(MM2_new,j,PN2);
            MMM_1=MM_1;
            MMM_2=MM_2;
            for k =1:f
                for h =1:c
                    if MM1_new(k,h) ~=MMM_1(k,h)
                        MM_2(k,h)=j;
                    end
                    if MM2_new(k,h)~=MMM_2(k,h)
                        MM_1(k,h)=number;
                    end
                end
            end
            %Update the variables after eliminating those points
            MM_new(1+f*(j-1):j*f,:) = MM_2;
            MM_new(1+f*(number-1):number*f,:) = MM_1;
            Lucho(1+f*(j-1):j*f,:)= MM_2;
            %Generate the plot Matrix
            for q=1:f
                for t=1:c
                    if MM_new(q+f*(j-1),t)==j
                        M_gen(q+f*(j-1),t)=MM_new(q+f*(j-1),t);
                    end
                end
            end
            for q=1:f
                for t=1:c
                    if MM_new(q+f*(number-1),t)==number
                         M_gen(q+f*(j-1),t)=number;
                    end
                end
            end
        end
end
%Generate the points that superpose
list = vertcat(list,superpose(f,c,M_gen,n));
if flag_d==true
    for l=1:f
        for y=1:c
            if MM_2(l,y)==0 &&MM_2_old(l,y)~=0
                list_2=vertcat(list_2,[l,y]);
            end
        end
    end
end
%Generate the points that have been free
if isempty(list_2)==0
    [W,~]=size(list_2);
    for b = 1:W
        M_fake(list_2(b,1),list_2(b,2))=100;
    end
```

```
end

if isempty(list)==0
    [T,~]=size(list);
    for z = 1:T
        M_fake(list(z,1),list(z,2))=100;
    end
end
%Generate obstacles
if isempty(obs)==0
    [o,~]=size(obs);
    for u=1:o
        M_fake(obs(u,1),obs(u,2))=1000;
    end
end
%Generate ocuppancy matrix and vector of positions
ON = ocupationN2(f,c,M_gen,n);
KN = letspaintN2(ON,f,c,M_gen,n);
[xN,yN] = drawingN(KN,n);
%Plot the final result
picaso_proces(xN,yN,P_tot,n,M_fake,colors,number,j)
pause(0.3)
%Update the variables that return
positionN=P_tot;
[~,DD] = distance_matrixN2(f,c,MM,n,positionN);
%Calculate the H for each region
H = calculusH(DD);
positionN=P_tot;
end
```

Important Functions

Create occupancy matrix

```
 function [MM] = redefineN(M,n)
%initialize varibale MM, which contains all sub matrix M of the map
[f,c] = size(M);
MM = zeros(f*n,c);
rowsMM=0;
%start the loop to create matrix MM
for i = 1:n
    for j = 1:f
        for k=1:c
            if M(j,k) ==i
                MM(j+rowsMM,k)=i;
            end
        end
    end
    rowsMM=i*f;
end
```

Find the center of an occupancy matrix

```
function [positionN] = optimalN2(m,n,MM,r)
%initialize variables
```

```
support = zeros(m,n);
[f,~]=size(MM);
positionN=[];
numero = 0;
clocking = 0;
%initialize loop
for l=1:f/r:f
    numero = numero+1;
    clocking = clocking+f/r;
    %Calculus of distance for each point
    for i =1:m
        for j =1:n
            support(i,j) = cost(MM(l:clocking,:),i,j,numero);
        end
    end
    %Find the minimun point
    [minimo, ~] = min(support(:));
    ListX = [];
    ListY = [];
    contador =0.0;
    X_new =0.00;
    Y_new =0.00;
    %Create a list of points with the possible centers,
    %Due to its a grid map can be between two points the center
    for i = 1:m
        for j =1:n
            if round(support(i,j),2) == round(minimo,2)
                contador = contador +1.0;
                ListX(contador) = [i];
                ListY(contador) = [j];

            end
            if and(i ==m,j ==n)
                for k = 1.0:contador
                    X_new = X_new+ListX(k);
                    Y_new =Y_new +ListY(k);
                end
            end
        end
    end
    %Calculate the center
    X_new = double(X_new/contador);
    Y_new = double(Y_new/contador);
    position = [X_new,Y_new];
    positionN=vertcat(positionN,position);
end
end
```

Calculus of distance matrix

```
function [D,DD] = distance_matrixN2(f,c,MM,n,PN)
%Initialize variables
D = [];
DD = [];
suport = zeros(f,c);
calculus = zeros(f,c);
indice =0;
%Calculus of the distance at each point to the center
for k = 1:n
    for i =1:f
        for j = 1:c
            calculus(i,j) = sqrt((i-PN(k,1))^2+(j-PN(k,2))^2);
```

```
            if k == MM(i+indice,j)
                suport(i,j) = calculus(i,j);
            else
            end
        end
    end
    indice =indice+f;
    DD = vertcat(DD,suport);
    D=vertcat(D,calculus);
    suport = zeros(f,c);
    calculus = zeros(f,c);
end

end
```

Exchange of territories

```
function [M_1,M_2] = DistributionN2(f,c,M_1,M_2,number1,number2,D1,D2,DD1,DD2)
%Exchanges territories between both robots
for i = 1:f
    for j = 1:c
        if D2(i,j)<=DD1(i,j)
            M_2(i,j) = number2;
            M_1(i,j) = 0;
        end
        if D1(i,j)<DD2(i,j)
            M_1(i,j) = number1;
            M_2(i,j) = 0;
        end
    end
end
end
```

Calculus of cost functional

```
function H = calculusH(Distance)
%Calculus of H for each robot
[f,c]=size(Distance);
H = 0;
for i = 1:f
    for j = 1:c
        H = H+Distance(i,j);

    end
end
end
```

Support functions

```
function [C] = contactN(M,n)
%Initialize variables
[f,c] = size(M);
MM = zeros(f+2,c+2);
MM(2:f+1,2:c+1)=M;
C = zeros(n,n);
%Find the neighbors
for i = 1:f
```

```
    for j = 1:c
        if MM(i+1,j+1)~= 1000 && MM(i+2,j+1)~=1000 && MM(i+1,j+1)~= 0 &&
MM(i+2,j+1)~=0
            if MM(i+1,j+1) ~= MM(i+2,j+1)
                C(MM(i+1,j+1),MM(i+2,j+1)) =1;
                C (MM(i+2,j+1),MM(i+1,j+1)) =1;
            end
        end
        if  MM(i+1,j+1)~=1000 && MM(i+1,j+2)~=1000 && MM(i+1,j+1)~= 0 &&
MM(i+1,j+2)~=0
            if MM(i+1,j+1) ~= MM(1+i,j+2)
                C(MM(1+i,1+j),MM(i+1,j+2)) = 1;
                C(MM(i+1,j+2),MM(i+1,j+1)) = 1;
            end
        end
    end
end
```

```
function [xN,yN] = drawingN(KN,n)
%initialize variables
[f,c]=size(KN);
xN=[];
yN=[];
row=f/n;
contador = 1;
indice=0;
%Generate a vector with the positions of the frontier
for k=1:n
    for i =1:row
        for j =1:c
            if KN(i+indice,j)==k
                xN(k,contador) = i;
                yN(k,contador) = j;
                contador = contador +1;
            end
        end
    end
    contador = 1;
    indice=indice+row;
end
end
```

```
function [KN]=letspaintN(ON,M,n)
%Initialiaze variables
[f,c] = size(M);
[h,t] = size(ON);
KN = zeros(h,t);
indice=0;
%Begin the loop and generate the matrix of frontier positions
for p=1:n
    for i =1:f-1
        for j =1:c
            if i ==1
                if ON(i+indice,j) ==1
                    KN(i+indice,j) =1;
                end
            end
```

```
            if j ==1
                if ON(i+indice,j) ==1
                    KN(i+indice,j) =1;
                end
            end
            if j ==c
                if ON(i+indice,j) ==1
                    KN(i+indice,j) =1;
                end
            end
            if and(ON(i+indice,j)==0, ON(i+indice+1,j) ==1) ||
and(ON(i+indice,j)==1,ON(i+1+indice,j)==0)
                KN(i+indice,j) = ON(i+indice,j);
                KN(i+1+indice,j) = ON(i+1+indice,j);
            end
        end
    end

    for i=1:f
        for j=1:c-1
            if i ==f
                if ON(i+indice,j) ==1
                    KN(i+indice,j) =1;
                end
            end
            if and(ON(i+indice,j)==0, ON(i+indice,j+1) ==1) ||
and(ON(i+indice,j)==1,ON(i+indice,j+1)==0)
                KN(i+indice,j) = ON(i+indice,j);
                KN(i+indice,j+1) = ON(i+indice,j+1);
            end
        end
    end
    if ON(f+indice,c) ==1
        KN(f+indice,c) =1;
    end
    for i =1:f
        for j =1:c
            if KN(i+indice,j) == 1
                KN(i+indice,j) =M(i,j);
            end
        end
    end
    indice=indice+f;
end
end
```

```
function [ON] = ocupationN(M,n)
%Initialize
[f,c] = size(M);
ON=zeros(f*n,c);
indice=0;
%Convert the occupancy matrix to a 1-0 matrix
for k=1:n
    for i = 1:f
        for j = 1:c
            if M(i,j) == k
                ON(i+indice,j) = 1;
            end
        end
```

```
    end
    indice=indice+f;
end
```

```
function [M] = recreation(MM_1,number,PN)
%initialize variables
[f,c] = size(MM_1);
MM = zeros(f+2,c+2);
PN=round(PN);
L = MM;
L(PN(1,1)+1,PN(1,2)+1)=number;
MM(2:f+1,2:c+1)=MM_1;
contador =0;
[f,c]=size(MM);
%Create a long loop
while contador <f*f
    contador =contador+1;
    %From center position reconstruct the matrix
    for i =1:f
        for j =1:c
            if L(i,j)== number
                if MM(i+1,j)==number
                    L(i+1,j)=number;
                end
                if MM(i-1,j)==number
                    L(i-1,j)=number;
                end
                if MM(i,j+1)==number
                    L(i,j+1)=number;
                end
                if MM(i,j-1) ==number
                    L(i,j-1)=number;
                end
            end
        end
    end

end
[f,c] = size(MM_1);
M=L(2:f+1,2:c+1);
end
```

```
function list = superpose(f,c,MM,n)
%initialize variable
list = [];
%initialize loop and find the points with superposition
for i =1:n
    if i <n
        for j = i+1:n
            for k =1:f
                for d=1:c
                    if MM(k+f*(i-1),d) ~= MM(k+f*(j-1),d)
                        list = vertcat(list,[k,d]);
                    end
                end
            end
        end
```

```
    end
end
end
```

```
function [D1,D2] = SearchDN2(D,i,j,n,f)
clocking = 1;
%Returns the D matrix for each robot
for p = 1:n
    if p == i
        D1 = D(clocking:f*p,:);
    end
    if p ==j
        D2 = D(clocking:f*p,:);
    end
    clocking = clocking+f;
end
```

Plot Functions

```
function []= picaso(xN,yN,PN,n,M_support,colors)
%Initialize variables
re = [];
ro = [];
[f,c]=size(M_support);
%Plot each matrix robot
for i=1:n
    re = horzcat(re,strcat('region ',int2str(i)));
    ro = horzcat(ro,strcat('robot',int2str(i)));
    scatter(xN(i,:),yN(i,:),150,colors(i,:),'filled','square','DisplayName',re)
    hold on
    axis([-2 12 -2 12])
    scatter(PN(i,1),PN(i,2),150,colors(i,:),'*','DisplayName',ro)
    legend('location','southwest','NumColumns',4)
    re = [];
    ro = [];
end
x_100 = [];
y_100 = [];
for i =1: f
    for j=1:c
        if M_support(i,j) ==100
            x_100 = horzcat(x_100,i);
            y_100 = horzcat(y_100,j);
        end
    end
end
%Find obstacles
x_1000 = [];
y_1000 = [];
for i =1: f
    for j=1:c
        if M_support(i,j) ==1000
            x_1000 = horzcat(x_1000,i);
            y_1000 = horzcat(y_1000,j);
        end
    end
end
```

```
%Plot sharing points and obstacles
if size(x_100)>0
    scatter(x_100,y_100,150,'cyan','filled','square','DisplayName','Shared
region')
end
if size(x_1000)>0
    scatter(x_1000,y_1000,150,'black','filled','square','DisplayName','Obstacle')
end
%Eliminate ploted points in (0,0)
axis([-2 f+2 -2 c+2])
scatter(0,0,150,'white','filled','square','DisplayName',' ')
hold off
pause(.6)
end
```

```
function []= picaso_proces(xN,yN,PN,n,M_support,colors,number,j)
%Initialize variables
re = [];
ro = [];
[f,c]=size(M_support);
%Plot each matrix of each robot
for i=1:n
    re = horzcat(re,strcat('region ',int2str(i)));
    ro = horzcat(ro,strcat('robot',int2str(i)));
    scatter(xN(i,:),yN(i,:),150,colors(i,:),'filled','square','DisplayName',re)
    hold on
    scatter(PN(i,1),PN(i,2),150,colors(i,:),'*','DisplayName',ro)
    legend('location','southwest','NumColumns',4)
    title(strcat(' Robot',strcat(int2str(number),strcat(' communicating '))));
    re = [];
    ro = [];
end
%find obstacles
x_100 = [];
y_100 = [];
for i =1: f
    for j=1:c
        if M_support(i,j) ==100
            x_100 = horzcat(x_100,i);
            y_100 = horzcat(y_100,j);
        end
    end
end
x_1000 = [];
y_1000 = [];
for i =1: f
    for j=1:c
        if M_support(i,j) ==1000
            x_1000 = horzcat(x_1000,i);
            y_1000 = horzcat(y_1000,j);
        end
    end
end
%plot shared points and obstacles
if size(x_100)>0
    scatter(x_100,y_100,150,'cyan','filled','square','DisplayName','Shared
region')
end
if size(x_1000)>0
    scatter(x_1000,y_1000,150,'black','filled','square','DisplayName','Obstacle')
```

```
end
%eliminate error points in (0,0)
axis([-2 f+2 -2 c+2])
scatter(0,0,150,'white','filled','square','DisplayName',' ')
hold off
pause(.6)
end
```